KU LEUVEN

ARENBERG DOCTORAL SCHOOL
FACULTY OF ENGINEERING TECHNOLOGY

IDENTIFYING AND AMPLIFYING NON-TECHNICAL SKILLS

IN SOFTWARE ENGINEERING EDUCATION

WOUTER GROENEVELD

DISSERTATION PRESENTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF ENGINEERING TECHNOLOGY

JULY 2023

This dissertation was presented in partial fulfilment of the requirements for the degree of Doctor of Engineering Technology (PhD).

Examination committee:

Prof. dr. Joost Vennekens, supervisor
Prof. dr. Peter Van den Broeck, chair
Prof. dr. Greet Langie
dr. ir. Ronny Martens
Prof. dr. Kris Aerts, co-supervisor
Prof. dr. Giovanni Samaey
Prof. dr. Renate Motschnig

ISBN 978 94 646 6895 7
NUR 123
KBR D/2023/Wouter Groeneveld, uitgever.

First printing: *July 2023*



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Charts generated with amCharts, Nodeplotlib, Matplotlib. Text processed with Sublime Text, Pandoc, MacTeX, Make. Data analyzed with Node, Python, R. Productivity fueled by Alfred, Navidrome, Mariage Frères tea.

PUBLISHED BY BRAIN BAKING, KU LEUVEN



Dedicated to all programmers

who struggle to find their creative groove.

Abstract

Software engineering is defined as the systematic appliance of engineering methods to software development. These engineering methods are without a doubt technical: software engineers daily have to juggle techniques such as continuous integration, deployment pipelines, microservices, horizontal and vertical slice architectures, automated testing, load balancing, and so forth.

As a result, in higher education, software engineering curricula are overflowing with technical courses to acquaint students with a multitude of engineering methods. And yet, according to industry experts, software engineering syllabi seem to be falling short of delivering great software developers. Several surveys, including our own, that ask professionals *"what makes a great software engineer?"* specify personal characteristics, communication, decision making, and creative problem solving next to technical knowledge as vital skills. It is clearly no longer sufficient to be technically proficient.

Computing accreditation programs for higher education struggle to follow suit. Even with the attempts of task forces and institutions to improve software engineering education, the gap between academia and industry persists. The demand for software development talent is much higher than the number of graduates universities can deliver each year, putting even more pressure on the skill gap problem.

In this dissertation, we identify what exactly those *non-technical* industrial requirements of modern software engineers are, how big

the gap between those requirements and software engineering education is, and how we as educators can amplify students' non-technical skills in order to reduce that gap. We first focus on non-technical skills in general before zooming in on *creativity* as one of the key attainable skills to becoming a great software engineer.

We present a framework for mastering seven creative problem-solving skills: *technical knowledge*, *communication*, *constraints*, *critical thinking*, *curiosity*, *creative state of mind*, and *creative techniques*. A self-assessment tool called the Creative Programming Problem Solving Test was derived from that to help gauge your current creativity mastery level in context of a software development project.

In addition, each chapter summarizes implications for educational practice. Finally, a practical approach to creativity in software development is explored, specifically geared towards software engineering professionals.

This work contributes to computing education research by helping shed light on the non-technical academia-industry skill gap and by bringing research on creativity from the field of cognitive psychology closer to the field of computing education.

Samenvatting

Software engineering wordt gedefiniëerd als de systematische toepassing van engineering methodes op softwareontwikkeling. Deze engineering methodes zijn zonder twijfel technisch: software-ingenieurs moeten dagelijks jongleren met technieken zoals *continuous integration*, *deployment pipelines*, *microservices*, *horizontal* en *vertical slice architectures*, *automated testing*, *load balancing*, enzovoort.

Als gevolg hiervan staan de leerplannen voor software engineering in hoger onderwijs vol met technische cursussen om studenten vertrouwd te maken met een groot aantal van deze engineering methodes. En toch beweren experten uit de industrie dat software engineering syllabi tekortschieten in het aanleveren van goede softwareontwikkelaars. Verschillende vragenlijsten, inclusief de onze, die professionals de vraag *"wat maakt een goede software-ingenieur?"* stellen, specifiëren persoonlijke karakteristieken, goede communicatie, besluitvorming, en het creatief oplossen van problemen naast technische kennis als belangrijke vaardigheden. Het volstaat duidelijk niet meer om enkel technisch vaardig te zijn.

Het hoger informaticaonderwijs heeft moeite om dit voorbeeld te volgen. Zelfs met de pogingen van taskforces en instellingen om het onderwijs in software engineering te verbeteren, blijft de kloof tussen de academische wereld en de industrie bestaan. De vraag naar talent voor softwareontwikkeling is veel groter dan het aantal afgestudeerden dat universiteiten elk jaar kunnen leveren, waardoor het probleem van de vaardigheidskloof nog meer onder druk komt

te staan.

In dit proefschrift identificeren we wat precies die *niet-technische* industriële vereisten van moderne software-ingenieurs zijn, hoe groot de kloof is tussen die vereisten en de software-ingenieursopleiding, en hoe wij als docenten de niet-technische vaardigheden van studenten kunnen versterken om die kloof te verkleinen. We richten ons eerst op niet-technische vaardigheden in het algemeen voordat we inzoomen op *creativiteit* als een van de belangrijkste aanleerbare vaardigheden om een goede software-ingenieur te worden.

We presenteren een raamwerk voor het beheersen van zeven creatieve probleemoplossende vaardigheden: *technical knowledge*, *communication*, *constraints*, *critical thinking*, *curiosity*, *creative state of mind*, en *creative techniques*. Een zelfevaluatietool genaamd de Creative Programming Problem Solving Test is daarvan afgeleid om uw huidige creativiteitsbeheersingsniveau te helpen peilen in de context van een softwareontwikkelingsproject.

Daarnaast vat elk hoofdstuk de implicaties voor de onderwijspraktijk samen. Ten slotte wordt een praktische benadering van creativiteit in softwareontwikkeling onderzocht, specifiek gericht op professionals in het werkveld.

Dit werk draagt bij aan onderzoek naar didactiek in de informatica door licht te werpen op de niet-technische vaardigheidskloof tussen de academische wereld en de industrie en door onderzoek naar creativiteit op het gebied van cognitieve psychologie dichter bij het gebied van informaticaonderwijs te brengen.

Preface

What is missing in software development teams these days? That's the million dollar question that got me into this mess. I've spent 11 years as a software engineer slowly but consistently shifting my interest from technicality and program architectures to the mysticism called non-technical coding skills.

When I got involved in coaching and the hiring process, I noticed a few odd things. Why was it that new recruits mostly caught up with our frameworks and best practices, but sometimes failed to grasp what really matters: jelling with the team and collaboratively solving problems? What does it mean to be a truly great programmer, besides the obvious technical mastery? What is missing indeed.

That question kept me up at night and ultimately lured me back to academia—first as a part-time teaching assistant, then as a full-time employee more or less evenly dividing my work between research and teaching. Five bedazzling years of research and teaching activities, `810` commits, `1020 MiB` of data, and `152.834` words later, I can finally say I better understand what makes a truly great programmer. A great software engineer doesn't limit their interests to compilers, design patterns, distributed systems, and deployment architectures—although, of course, they're all vital. Great software engineers also teach themselves how to effectively communicate and collaborate, how cognition in context of programming works, how to be critical, and perhaps most importantly: how to be a *creative* programmer.

In an attempt to break free of the conventional academic

boundaries, a practical summary of the most important findings of this dissertation has been published with Manning as *The Creative Programmer* (Groeneveld, 2023)[1]. With this dissertation and the practical handbook, I hope to inspire both academics and industry professionals to look beyond segregated technical requirements.

[1] An overview of the handbook is provided in the conclusion section of this work, on page 263.

This work also contributes to closing the gap between academia and industry. As an academic researcher, I did my best to end every publication with implications for education. As a teaching assistant, I did my best to inject our findings directly into the teaching material of our local faculty. As an industry expert, I did my best to share my knowledge to hopefully spark the interest of other programmers, so that they in turn can spread the word.

It's funny to look back and see how the original idea for a doctorate evolved:

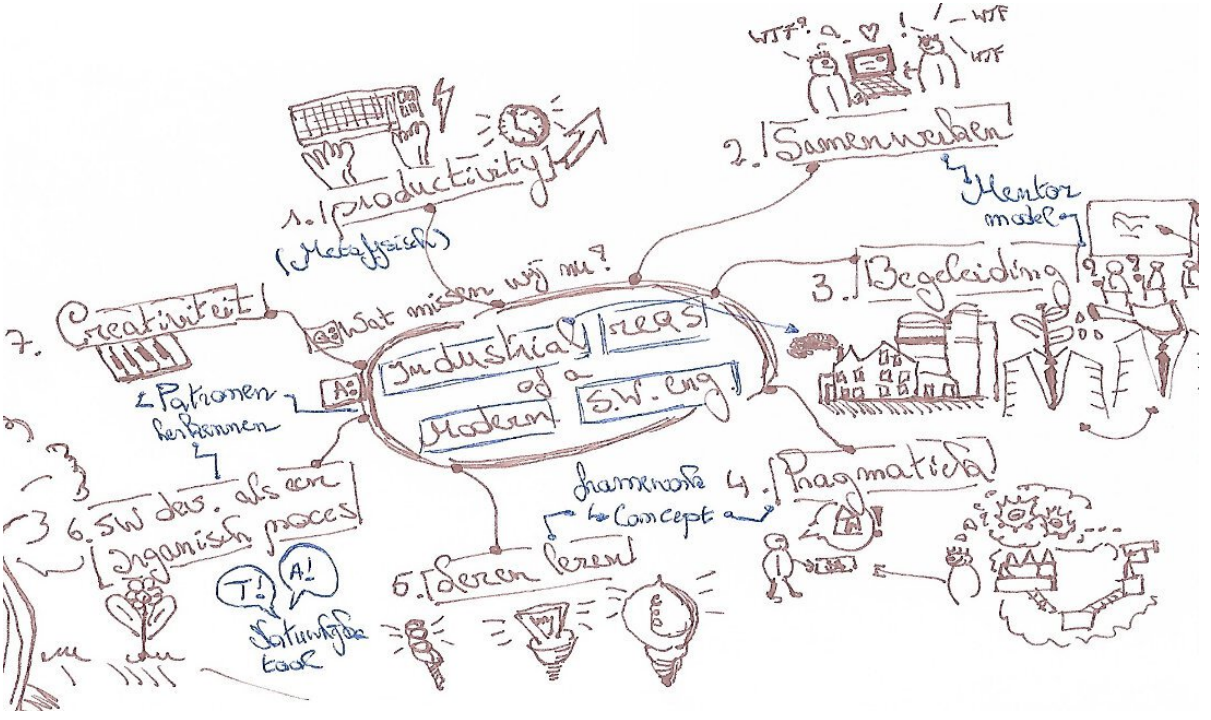


Figure 1: An early sketch of the PhD proposal from 06/2018. The full original concept is available at https://brainbaking.com/post/2018/06/phd-proposal/.

The original title, *Bridging the gap between software development requirements in the industry and educations in the academia* was

thankfully adapted to a more digestible *Identifying and amplifying non-technical skills in software engineering education*.

As the early sketch of Figure 1 shows, I had the grand idea of answering the "what is missing" question by merging concepts from productivity, collaboration, coaching, pragmatism, lifelong learning, organic software development, and creativity. Then I got the wise advice to "focus, focus, focus". Still, it's interesting to see those concepts never really floating too far away—as you will soon discover yourself, I might have gone with exactly that.

My failed attempts to tackle a PhD like an agile software development process[2] made it painfully clear that academia and industry are indeed miles apart. I won't lie: it's been hard. In fact, it's been *very* hard—not the work itself, but the persistent loneliness that accompanies academic research. A friend correctly noted that "research is awesome but lonely". As a software engineer who is used to pair programming all day and very closely collaborating with dozens of people, I was clearly ill-prepared. The global pandemic that emerged in 2020 didn't exactly make this any easier.

[2] See https://brainbaking.com/post/2018/10/lean-doctorate/.

I prevailed. Even as this body of work can be considered finished, I'm not done talking about it. I also regularly write about this subject at https://brainbaking.com/. If you happened to stumble upon this text and would like to get in touch, by all means, I'd love to talk to you! My website contains my coordinates.

Thanks for reading,

Wouter.

Acknowledgements

Any substantial work such as this one stands on the shoulders of giants. You probably won't be reading this if it weren't for the **612 people** (!) that all actively contributed in some way by either providing the data or helping to analyze it[3]: thank you very much!

To my supervisor Joost Vennekens and co-supervisor Kris Aerts; for giving me the freedom to pursue very vague ideas and even making sure the at first seemingly fruitless hunt was financed.

To my other paper co-authors Hans Jacobs, Brett A. Becker, Laurens Luyten, Dries Martin, Tibo Poncelet, Lynn Van den Broeck, and Arne Duyver; for helping spread the word and putting me in contact with like-minded folks.

To the members of the supervisor and examination committee; for putting in the time to critically evaluate this work and assisting in shaving off its rough edges.

To the LESEC and CS OVI[4] research (sub)groups at KU Leuven; for discussing unrelated literature that sometimes inspired me to link back to my own work and for at least pretending to patiently listen to my software engineering ramblings.

To all research participants; for their creative discussions, insights, and suggestions that make up for most of this large body of work. We wouldn't have anything to analyze or report without your input.

To my students at Diepenbeek Campus over the last five academic years; for making teaching a challenge without making my life impossible, leaving me with ample energy left to write.

To the computing education research community; for laying the

[3] If I had to include passive contributors—authors of cited work—that number would go up by another 697.

[4] Say hi at `https://set.kuleuven.be/LESEC` and `https://wms.cs.kuleuven.be/cs/english/research` respectively.

groundwork, reviewing and most of the time graciously accepting our contributions, and for providing welcoming guidance.

To all the folks at Manning; for recognizing the potential of the first rough manuscript of *The Creative Programmer* and helping me shape and promote it, which I consider to be an inseparable part of this work.

Lastly, and probably the person I owe the most to, to my wife Kristien Thoelen; for having to listen to my all too frequent whining and gripes about life in academia and the silliness that frequently comes with it.

Contents

Abstract **1**

Samenvatting **3**

Preface **5**

Acknowledgements **9**

1 Introduction **15**
1.1 The structure of this thesis 18
1.2 The scope of this work 19

I Identifying Non-technical Skills **23**

2 Defining Non-technical Skills **27**
2.1 Skill, knowledge, ability, or competency? 28
2.2 Soft skills, people skills, or professional skills? 32

3 Teaching Trends: Literature Review **37**
3.1 Related work 38
3.2 The review process 39
3.3 Results & discussion 40
3.4 Limitations 50
3.5 Conclusion 52

4 Non-technical Industry Requirements **53**
4.1 Related Work 54
4.2 The Delphi process 55

4.3 Results & discussion . . . 58
4.4 Threats to validity . . . 67
4.5 Conclusion . . . 68

II The Academia-Industry Skills Gap 71

5 Teaching Trends: Program Syllabi 75
5.1 Collecting syllabi data . . . 76
5.2 Results & discussion . . . 80
5.3 Threats to validity . . . 84
5.4 Conclusion . . . 85

6 Identifying the Skill Gap 87
6.1 Related work . . . 88
6.2 Methodology . . . 90
6.3 Gap analysis results . . . 96
6.4 Interview results . . . 99
6.5 Threats to validity . . . 108
6.6 Conclusion . . . 110

III Creativity as a Non-technical Skill 113

7 Defining Creativity 117
7.1 The psychology of creativity . . . 117
7.2 Creativity and design . . . 121
7.3 Creativity and engineering . . . 123
7.4 Conclusion . . . 124

8 Teaching Trends: Creativity 127
8.1 The review process . . . 129
8.2 Results & discussion . . . 131
8.3 Approaches, tools, & pedagogical implications . . . 139
8.4 Conclusion . . . 143

9 Professionals' Perceptions of Creativity 145
9.1 The focus group procedure . . . 146
9.2 Results & discussion . . . 149
9.3 Limitations . . . 160

9.4 Conclusion 161

10 Students' Perceptions of Creativity **163**
10.1 Perceiving (differences in) creativity 164
10.2 Methodology 165
10.3 Bottom-up results 169
10.4 Top-down results 175
10.5 Implications for educational practice 183
10.6 Limitations 184
10.7 Conclusion 185

11 Measuring Creativity **187**
11.1 Background 188
11.2 Survey design 190
11.3 Results & discussion 197
11.4 Limitations 202
11.5 Conclusion 203

IV Amplifying Creative Skills **205**

12 Creativity & Motivation **209**
12.1 Sparking creativity with the GBA 210
12.2 Engaging students in grading 212
12.3 Limitations 222
12.4 Conclusion 222

13 Creativity & Code Quality **225**
13.1 Existing work 226
13.2 Evaluating code quality & creativity 228
13.3 Results & discussion 234
13.4 Limitations 239
13.5 Conclusion 240

14 Interdisciplinary Creativity **241**
14.1 Background 243
14.2 Methodology 245
14.3 Results & discussion 249
14.4 Limitations 256

14.5 Conclusion 256

A Conclusion **259**
A.1 What are non-technical requirements? 259
A.2 How big is the industry-academia gap? 260
A.3 How to improve education to reduce that gap? 261
A.4 Creative programming in practice 263
A.5 Suggestions for curriculum design 266
A.6 Suggestions for future research 269

List of Figures **271**

List of Tables **272**

List of Acronyms **275**

1 Introduction

The IEEE Standard Glossary of Software Engineering Terminology defines Software Engineering (SE) as "the systematic appliance of engineering methods to software development" (IEEE Computer Society, 1990). What exactly these methods are supposed to be is left to the interpretation of the engineer. Requirements analysis, design, measurement, deployment, integration, and maintenance are the usual suspects here. These methods share a common theme: they are all technical. A software engineer is without a doubt a highly technical profession.

As a result, SE education focuses almost entirely on technicality: our analysis[1] shows that SE syllabi in higher education consist of segregated and highly specialized courses with the purpose of gradually teaching the student slices of technicality. Networking, software architectures, operating systems, neural networks, object-oriented software development, ... The constant and rapid evolution of hardware and software leaves little wiggle room for other courses.

[1] See Chapter 5.

And yet, according to industry experts, these SE syllabi seem to be falling short of delivering great software developers. Several surveys, including our own, that ask professionals *"What makes a great software engineer?"* identify personal characteristics, communication, decision making, and creative problem solving next to technical knowledge as important attributes (Li, Ko, and Zhu, 2015). Garousi, Giray, Tüzün, Catal, and Felderer (2018) have identified a gap between SE education and industry needs, and claim it is no longer sufficient to be technically proficient.

Computing accreditation programs for higher education struggle to follow suit. The ACM/IEEE Accreditation Board for Engineering and Technology (ABET) project includes a *Professional Practices* category since 2001 as one of its 11 principles:

> *CC2001 must include professional practice as an integral component of the undergraduate curriculum. These practices encompass a wide range of activities including management, ethics and values, written and oral communication, working as part of a team, and remaining current in a rapidly changing discipline* (*The IEEE CS/ACM Joint Task Force on Computing Curricula, 2001, p 13*).

CS2008, which was an update of CC2001, found that:

> *Communication and other transferable skills had to be addressed* (*The IEEE CS/ACM Joint Task Force on Computing Curricula, 2008, p 11*).

However, only one unit was added with relatively little refinement in this direction. It took until CS2013 to explicitly mention *soft skills* next to the importance of social issues and professional practice (The IEEE CS/ACM Joint Task Force on Computing Curricula, 2013). These guidelines removed several old units and added Professional Communication and Sustainability. ABET specifies only five "Student Outcomes" in their Criteria for Accrediting Computing Programs 2019–2020 (ABET, 2018), of which three can be considered non-technical skills:

- Communicate effectively in a variety of professional contexts;
- Recognise professional responsibilities and make informed judgements in computing practice based on legal and ethical principles;
- Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.

As we will discover in Chapter 5, educational institutions have responded by integrating some of these required skills into curricula. However, unlike more technical courses, the structuring and inclusion of non-technical courses is not very uniform between institutions. For instance, there is no CS1 for soft skills (Becker and Quille, 2019).

Still, academics acknowledge the growing urgency of competences beyond technicalities. For instance, Schumm, Joseph, Schroll-Decker, Niemetz, and Mottok (2012) indicate the rising importance of teaching other competences in addition to technical knowledge:

> *The professional requirements in software engineering have become highly volatile due to the complexities of project development and rapid and innovative changes occurring in the field. Therefore, the development of inter-personal and social competences has gained central importance in the training of software developers.*

Even with the attempts of task forces and institutions to improve SE education, the gap between academia and industry persists. Our syllabi analysis from Chapter 5 shows that computing and SE courses still neglect non-technical skills in the curriculum, and interviews with professionals from chapter Chapter 6 and onward reveal that education still falls short of expectations.

These issues related to SE education led to the following overarching PhD Research Questions (RQs):

PhD Research Questions

1. *What exactly are the non-technical industrial requirements of modern software engineers?*
2. *How big is the gap between those requirements and software engineering education?*
3. *How can we improve upon education to reduce that gap?*

Each part of this work strives to answer one of these questions.

The questions we ask are more relevant than ever: the amount of IT-related job offerings is staggering and will only rise in the near future. As Andreessen (2011) said, "*Software Is Eating the World*". Andreessen's statement is over a decade old and has only grown in pertinence.

The demand for talent is much higher than the number of graduates universities can deliver each year, putting even more pressure on the skill gap problem (Wilson, 2017). Industry tries to solve this by creating initiatives such as the coding boot-camp. Companies are willing to hire people who showcase non-technical skills, regardless of their technical knowledge:

> *The need for these non-technical skills is so great that some IT companies indicate that they will hire individuals with minimum technical skills so long as they demonstrate solid soft and business skills* (*Bailey, 2014*).

Technical knowledge is simply "patched on" by using coding camps or on-the-job learning. Unfortunately, these camps leave much to be desired in terms of delivering resilient graduates.

When it comes to educational attention in the world of computing, non-technical skills are still shoved aside or viewed as less relevant than technicality and knowledge. Yet these skills are becoming more and more important in the industry because of the way software is created: together, in close collaboration (Capretz and Ahmed, 2018; Matturro, Raschetti, and Fontán, 2015).

We hope that answering the above RQs helps contributing to closing the skill gap by gently putting non-technical skills where they belong—in the spotlights.

1.1 The structure of this thesis

In Part I, we first untangle the meaning of the term "non-technical" and the often confusing synonyms such as the aforementioned "soft skills" (Chapter 2), and provide an overview of current teaching trends (Chapter 3). Then, a Delphi study is presented to help identify non-technical skills (Chapter 4) and answer `RQ1`.

In Part II, after discussing trends in European SE program syllabi (Chapter 5), a skill gap analysis is presented to help answering `RQ2` (Chapter 6). The gap analysis will identify creativity as an important non-technical skill that needs more attention in education.

In Part III, creativity is further explored in context of SE. First, possible definitions of the concept are unfolded and discussed in Chapter 7. Then, current teaching trends are presented in Chapter 8, professionals' perceptions of creativity (Chapter 9) and students' perceptions of creativity (Chapter 10) are compared, and a proposition to measure creativity in SE education is made (Chapter 11).

In Part IV, as a first step towards answering RQ3, we apply our new knowledge of creativity to introduce it in SE education. Chapter 12 discusses the influence of motivation on creativity using a case study with peer grading, and Chapter 13 inspects the influence of creativity on code quality, while Chapter 14 reports on a interdisciplinary intervention in an attempt to amplify creative problem solving.

Finally, The last chapter concludes this work.

Unless otherwise stated, each chapter corresponds to one of our publications, as listed in the epigraph.

1.2 The scope of this work

As the title suggests, we have focused our efforts on identifying skills for SE and improving SE education. However, depending on the university, it is not always clear what the boundaries of a typical SE grade are. For some institutions, SE is an optional part of the more theoretical grade called Computer Science (CS), while others offer an electrical engineering grade where SE is also touched upon. The ACM organization therefore recommends the usage of the broader term *computing* instead of CS as not to exclude more practical or engineering-based institutions.

To make matters even more confusing, in industry—and in academic papers—the term *software engineer* is sometimes interchanged with *software developer* or *programmer*. HR departments love to add the term "engineer" to job ads, even though it can be debated that a software engineer is not recognized as a true engineer[2].

For this work, the starting point has always been enterprise[3] software development: the creation of large pieces of software that call for multiple teams and multiple "systematic appliances of engineering methods", as IEEE mentioned. We are more interested in the various non-technical skills that are required in a practical process of building, maintaining, and deploying software in teams than in the scientific knowledge that is required in many more theoretical CS educations.

[2] For an extended discussion, see "Why Are Software Engineers (Not) Engineers" at https://brainbaking.com/post/2022/10/why-are-software-engineers-engineers/.

[3] Software that satisfies the needs of an organization is usually part of a larger system.

Of course, many software engineers—myself included—earn a CS degree before ending up in industry and contributing to enterprise software development, hence the vague borders. But not every software engineer attended a university: the majority of my ex-colleagues earned a bachelor's degree in informatics.

Therefore, in order to limit the scope and better focus our research, we did our best to adhere to the following rules:

- When conducting interviews with professionals, seek out team members who are concerned with the development of large pieces of software and who come into daily contact with code—independent of their current job title.
- When reviewing literature, focus on the practical approach of SE and problem solving in context of programming instead of the more theoretical CS publications.
- When conducting interviews with students or analyzing course syllabi, do not limit the search to universities that offer SE grades but consider including broader computing educations before zooming into SE[4].

[4] When interviewing professionals, we did not exclude candidates without a university degree, as beyond the academic borders that difference in grade matters less as work experience grows.

Our Systematic Literature Reviews (SLRs) always included synonyms when searching for related material. For example, when searching for "software engineering", we have also searched for "software development" or "software programming". These synonyms will be listed in each methodology section of the respective chapters.

Also, as mentioned before, SE is much broader than programming (risk analysis, maintenance, continuous deployment pipeline design, juggling conflicting wishes of clients, ...), but in light of this work, when it comes to zooming in on non-technical skills, the context will be problem solving with the help of code.

Perhaps not coincidentally, the master's degree of Engineering Technology our local faculty offers is a perfect fit: it sits comfortably between the theoretical CS degrees and the applied bachelor's degrees of colleges. When studying Engineering Technology, you can major in Electronics and ICT (part hardware, part software) or Informatics (more focus on software)—both majors are excellent fits for our SE profile.

KU Leuven describes an Engineering Science programme—the theoretical approach that leads to a CS degree—as follows[5]:

The Faculty of Engineering Science offers study programmes that are based on fundamental sciences: mainly physics and mathematics followed by engineering courses. The study programmes train students to design and optimise concepts for new products, processes, systems or services. These study programmes pay considerable attention to both conceptual aspects and methodological approaches and focus on the medium-term horizon.

[5] See https://www.esat.kuleuven.be/english/education/master/electrical-engineering/programme/engineering-science-vs-engineering-technology.

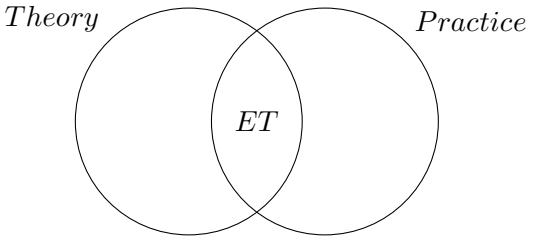


Figure 1.1: The Engineering Technology (ET) programme can be positioned between theoretical approaches such as the Engineering Science programme and practice-only approaches such as professional bachelor's degrees.

While the Engineering Technology programme is described as:

The Faculty of Engineering Technology trains students to become experts in developing, applying, implementing and improving products, processes and systems in specific applications. The programme is oriented towards practical applications and immediate usability in the professional field.

All educational interventions were carried out as part of the faculty of Engineering Technology at KU Leuven. The coming chapters describe additional context and limitations in scope for each individually carried out research.

Part I

Identifying Non-technical Skills

In this part, we try to answer the **first RQ** of this work:

PhD Research Question 1

What exactly are the non-technical industrial requirements of modern software engineers?

In order to answer this question, we first provide context for the term *non-technical skills* in Chapter 2. We then turn to both academia and industry. For the former, we performed a SLR (Chapter 3), and for the latter, we we conducted a Delphi study with industry experts (Chapter 4). Both findings have been peer reviewed and published as Groeneveld, Vennekens, and Aerts (2019) and Groeneveld, Jacobs, Vennekens, and Aerts (2020b) respectively.

These chapters shed light on what industry experts think are important non-technical skills for software engineers to succeed, and which non-technical skills are currently being taught in computing higher education. This gives us enough data to confidently answer `RQ1`.

2 *Defining Non-technical Skills*

What exactly are non-technical skills, and why are they called like that? Our own interpretation of this term was the result of an iterative process—in the past, we have utilized other terms to describe the same concept before arriving at *non-technical skill*. But first, let us take a look at what the literature says.

When sorting through papers, one often comes across recurring terms such as *human factors*, *required skills*, *soft skills*, ... Each of these, according to the author in question, has its own interpretation. This is very confusing, as also noted by Matturro et al. (2015): for example, *knowledge deficiency* does not always imply a deficiency in knowledge; it depends on the context of the paper.

Although they have been used as synonyms in literature, we have found the following frequently used terms that slightly differ in meaning: these terms range from *soft skills*, *21st century skills*, *intangible skills*, *human factors*, *interpersonal skills* and *generic competencies* to *social & emotional intelligence* and *people skills*. The problem with usage of different terms, or even the same terms, is that there is no single clear definition available. Each author uses their own interpretation, making the comparison of papers quite difficult. In an effort to clarify this for the purpose of this dissertation, we provide an overview of frequently encountered key terms and their definitions.

2.1 Skill, knowledge, ability, or competency?

Before venturing into the soft skill vs 21st century skill debate, we first have to determine whether or not the term *skill* is the appropriate one to use. The following subsections discuss similarities and differences between skill, knowledge, ability, and competency.

Skills

A skill, according to the Oxford Dictionary (Stevenson, 2010), is defined as:

> *The ability to do something well; expertise.*

Different persons showcase different levels of expertise. The Dreyfus model of skill acquisition (Dreyfus and Dreyfus, 1980) shows how and at which experience level these skills are, evolving with ample experience, from novice to mastery, as visible in Figure 2.1.



Figure 2.1: The Dreyfus model of skill acquisition. The closer to the top of the pyramid, the more intuitive you use the skill instead of sticking to the rules. A true expert grows meta cognitive abilities.

The term *lifelong learning skills* is often used loosely to identify a set of skills, abilities, or competencies. For example, Mouros (2003)

describes these as a set of skills included in many engineering programs, although they are sometimes also labeled as a set of competencies (Martinez-Mediano and Lord, 2012). To make matters worse, Martinez-Mediano and Lord use both *lifelong learning skills* and *lifelong learning competencies* interchangeably throughout their text.

Dunlap and Grabinger (2003) suggest that lifelong learning skills also involve much more than simply the ability to do something well. Their definition:

> *Intentional learning that people engage in throughout their lives for personal and professional fulfillment and to improve the quality of their lives.*

Another interesting set of skills related to this work is Sedelmaier and Landes's (2014b) Software Engineering Body Of Skills (SWEBOS), the counterpart of the conventional Software Engineering Body Of Knowledge (SWEBOK) (Bourque, Fairley, et al., 2014) that came to life because of SWEBOKs distinctive lack of non-technical skills. SWEBOS is divided into three hierarchical structures where competencies lay at the base for specialized skills. Again, *skill* is used in a broader sense.

The European Skills Competences qualifications and Occupations (ESCO) tries to make sense of all this confusion by grouping knowledge, ability, and competencies together in one big pillar called *skills* (Méhaut and Winch, 2012).

Knowledge

Fortunately, there is little contradiction to be found in the literature when it comes to the term knowledge—it is almost exclusively used to identify technicality. For example, the aforementioned SWEBOK guidelines describe generally accepted knowledge on SE in 15 different areas called *Knowledge Areas* (software requirements, software maintenance, software configuration management, systems engineering, computer engineering, ...). The emphasis is clearly on (technical) knowledge here. We agree and also view knowledge as the understanding of the scientific and technical side of, in our context, computing.

Knowledge can be gained in two ways: formally through study

or informally through implicit experience. The latter is called *tacit knowledge* by Senker (1995) and defined as:

> *The knowledge of techniques, methods and designs that work in certain ways and with certain consequences, even when one cannot explain exactly why.*

This matches more closely with ability than with traditional technical knowledge. Some examples of tacit knowledge mentioned by Heiberg Engel (2008) are playing the piano and bicycling: tacit knowledge is hard to put in words. Note that the line is very thin here: playing the piano obviously also requires technical knowledge.

As is the case in SWEBOK, most SE literature we encountered interpret knowledge as explicit or formal knowledge instead of tacit. However, the term *knowledge deficiency* sometimes refers to a lack of not only knowledge, but also skill and ability. Radermacher and Walia (2013) refer to knowledge deficiencies as "any skill, ability or knowledge of concept which a recently graduated student lacks based on the expectations of industry or academia". Their broad interpretation of a knowledge deficiency is more generally called a *skill gap* and is covered in Chapter 6.

Abilities

Ability is often seen as a determinant for successfully gathering knowledge and skills. Howe, Davidson, and Sloboda (1998) split abilities into innate and learned abilities, where innate abilities are called *talent*. The nature-nurture debate of talent vs ability is the central topic of research on genetic biases and achievement, where some scholars claim that talent does not exist while others consider the ignoring of well-established findings in behavior genetics to be a big mistake (Simonton, 2017).

To add to the confusion, in computing education literature, the term ability is frequently interchanged with the term skill. For example, Bennedssen and Caspersen (2008) question whether or not abstraction ability is an indicator for success for learning computer science, yet in their text, they interchange "abstraction ability" with "abstractive skills" nine times. The authors are very thorough in defining "abstraction" related to computing, citing

more than ten works while doing so, but at the same time forget to define or position the terms ability and skill.

Since we've encountered many works where an ability is defined as a way to gather (technical) knowledge as well as skills, the term is too broad for us to be employed to answer the research questions of this dissertation, as we are more interested in the "skill" part than in the "knowledge" part.

Competencies

According to Sedelmaier and Landes (2014a), the term *competence*[1] has gained a lot of popularity in the 21st century, but at the same time it is one of the most confusing terms. In SWEBOS, competencies are defined at the intersection between "factural knowledge" and soft skills.

Our literature study indicates that in context of work expectations, combinations of knowledge and skills are generally accepted to be called competencies. The Organisation for Economic Co-operation and Development (OECD) educational framework of Figure 2.2 summarizes the relation between skills, knowledge, and abilities.

[1] In this work, we use the term *competencies* as the required knowledge, skills, experiences, and attributes to carry out a defined function. A *competence* is the capability to carry out the function, while a *competency* is used in work context to describe this. As we are interested in required skills in industry, we use *competency* in work context.



Figure 2.2: The OECD education 2030 framework defines competencies as knowledge, skills, attitudes and values that lead to action. Image copyright OECD.

OECD (2018) defines a competency as "the ability to mobilize knowledge, skills, attitudes and values, alongside a reflective approach to the processes of learning, in order to engage with and act in the world"[2]. Competencies are more comprehensive than skills according to OECD. If we zoom in on the terms used in

[2] Note that OECD merges knowledge, skills, and attitudes into competencies, while ESCO groups knowledge, abilities and competencies into skills.

Figure 2.2, we again see that knowledge equals explicit technical knowledge (*disciplinary*, *interdisciplinary*, and *practical knowledge*).

Some researchers make the distinction between *generic* and *specific* competencies. For example, clearly communicating can be used across jobs and sectors, making it a generic competency. Sedelmaier and Landes (2014a) describe these as "generic soft skills".

In context of our RQs and to better focus this research, we are most interested in the skill tree: *cognitive and meta-cognitive*, *social and emotional*, and *physical and practical skills*. Therefore, we will stick with the term *skill*.

2.2 Soft skills, people skills, or professional skills?

NOW THAT WE HAVE settled with the term *skill*, another problem arises: are these people skills, soft skills, or perhaps 21st century skills? And what about professional skills? One thing is clear: we are looking beyond technical skills.

Hard and soft skills

Hard skills are typically associated with cognition: they are very specific and easy to assess. A quick scan through job ads teaches us that hard skills are either technical skills needed to perform a certain job (video editing, Java software development, ...) or more general but still specific (project management or other administrative roles).

What is more interesting is the opposite: *soft skills*, an umbrella term for a subset of different skills beyond the technical. SWEBOS distinguishes three categorical blocks for competencies for SE: *context-sensitive soft skills*, *generic soft skills*, and *factual* or technical knowledge. Some researchers come up with even more objectives such as *intangible soft skills*.

Unfortunately, when educators are trying to discern the importance of hard and soft skills, the vagueness of the concept once again stands in the way. Matteson, Anderson, and Boyden (2016) stated:

> *We universally recognise that soft skills are important, but when pressed to describe particular soft skills, the concept becomes murky.*

Some researchers strictly adhere to the Oxford Dictionary definition of a soft skill that emphasizes interaction with others:

> *Personal attributes that enable someone to interact effectively and harmoniously with other people.*

In our own work, we had originally defined a soft skill as "a non-technical skill that can be learned in a traditional academic setting and is relevant to proficiency in a given field of study, which may itself be largely technical" (Groeneveld, Becker, and Vennekens, 2020a)[3], interpreting the skills more broadly compared to the OECD definition. While we view soft skills as a synonym for non-technical skills, others do not. This ultimately made us abandon the term in more recent publications.

[3] See Chapter 5 for more information about the paper.

Human factors and people skills

Soft skills are frequently but not always exclusively associated with skills that stress social interaction or *people skills*. Especially in modern SE work environments, interacting well with teammates is crucial to the jelling of the team and the success of the software.

Capretz (2014) says that "SE is essentially a human activity. [...] Team members should also recognize that software development is a socio-technical practice". They use the term *human factors* to indicate that the overall system performance is dependent on the interactions of humans and other elements of the system. A frequently mentioned subset of these skills includes empathy, communication, openness, collaboration, and leadership.

While we feel that people skills are important, by excluding other important skills related to being a great software developer, the term itself is not a great fit for the purposes of this research. The same is true for *intrapersonal* and *interpersonal* skills which further break up communication-related skills.

Professional skills and competencies

Some papers use the term *professional skills* as the capability of adapting to change and ever-shifting work requirements (Shuman,

Besterfield-Sacre, and McGourty, 2005), while others combine the industry-focused "professional" with the comprehensive "competency" into *professional competencies* (Craps, Pinxten, Knipprath, and Langie, 2022).

Professional skills and competencies both hint at work requirements, sometimes also used in on the job training. The ABET professional skills list includes the technical knowledge required to perform well at at a certain job, and many HR training websites such as Training Express divide professional skills into hard and soft skills[4]. Therefore, we deem these terms too broad for our research.

[4] See https://www.trainingexpress.org.uk/professional-skills/.

The alternatives

In Groeneveld et al. (2019), we tried sidestepping the problem by using synonyms in the SLR query and using "beyond the technical" in the paper title. A year later, in Groeneveld et al. (2020b), we started out with *non-cognitive abilities* and provided the following definitions to get the conversation going:

- *abilities* as widely applicable competences, attitudes and behaviours which enable one to successfully achieve a goal.
- *non-cognitive abilities* as the subset of abilities that are not related to technical skills, i.e. excluding topics such as knowledge of programming languages or proficiency in IDEs.

Unsure about the correct usage of the terminology, we also asked the interviewed industry experts what they think is a more appropriate term. The following suggestions were made:

- Software Engineering Mindset
- Humane Software Development Skills
- Interpersonal Skills
- Engineering Skills
- Soft Skills (2x)
- Professional Attitudes
- Behavioral Skills
- Software Engineering Specific Non-technical Competences
- Non-technical Skills (3x)

- Transferable skills

Furthermore, a reprimand was made against our initial wording, again using *non-technical* in their response:

> *Non-cognitive seems not only incorrect but derogatory. All non-technical skills are cognitive, require practice, consideration and deliberate exercise to improve.*

Since *non-cognitive* is clearly the wrong word and *non-technical skills* is the most popular identification for the group of skills discussed in that paper[5], we will stick to that term for the remainder of this dissertation.

[5] Chapter 4 discusses this paper and the applied methodology in detail.

3 Teaching Trends: Literature Review

Publication details

This chapter has been published in *Proceedings of the 47th SEFI Annual Conference 2019* as *Software Engineering Education Beyond the Technical: A Systematic Literature Review* (Groeneveld et al., 2019).
Its main contributions are providing novel insights on relationships between non-technical skills and teaching aspects of SE curricula that can also serve as a springboard for further investigation of certain undervalued skills.

How should higher education shape the future of great software developers beyond technical knowledge? The need to invest into various skills like *creativity* and *collaboration* is not new, and not limited to SE alone. Papers ranging from 1994 (Friedman and Kahn, 1994) to 2018 (Capretz and Ahmed, 2018) share the global message to promote skills beyond the technical that still does not seem to be fully carried out by higher education. Multiple efforts have been made to renew curricula, but there is still no general consensus reached on the recognition and teaching of these skills.

As a first step towards identifying which direction SE education has taken to mitigate the skill gap introduced in Chapter 1, this chapter presents the results of a SLR focusing on the teaching of

non-technical skills[1]. The formalized process as a systematic review has proven to be very insightful for identifying the current state-of-the-art on a given subject. The method has been used widely in different fields, including SE research (Stapić, López, Cabot, de Marcos Ortega, and Strahonja, 2012). This chapter explores the current state-of-the-art of teaching non-technical abilities in SE education.

We first provide a bit of context by describing related work before moving on to the specifics of the methodology and the presentation and discussion of the results.

[1] As discussed in Chapter 2, in the 2019 SLR paper, the terms *non-cognitive* and *soft skills* were yet to be replaced by *non-technical skills*.

3.1 Related work

A LARGE AMOUNT OF research on non-technical skills for SE exists, including specific industry studies (Sedelmaier and Landes, 2014b; Stevens and Norman, 2016). However, most of these studies do not explicitly focus on the education system itself. They highlight shortcomings from the point of view of the industry with the help of, among other things, job ad analysis and focus groups, such as our own in the next chapter.

Examples of recent literature reviews similar to ours are Radermacher and Walia (2013) and Garousi et al. (2018). Although they do not focus soley on education (`"education"` is an optional string in their search queries), they provide similar insightful results. Other publications delve deeper into specific subjects, such as the impact of pair programming (Chen and Rea, 2018; Schumm et al., 2012) and the added benefit of improved confidence and self-esteem.

Lenberg, Feldt, and Wallgren (2015) conducted an interdisciplinary research of "*the psychology of programming*", redefining and reviewing behavioral SE. This combination of practical psychology with SE yields promising results for understanding what makes great developers tick (Graziotin, Wang, and Abrahamsson, 2015).

Another approach to identify skills is by investigating success stories in software development. Dutra, Prikladnicki, and França

(2015) explored high performance teams using the SLR method, while Li et al. (2015) simply asked practitioners: *"what makes a great software engineer?"* Unsurprisingly, more than 50% of the answers can be categorized as non-technical, attributed to external (teammates) and internal (personal characteristics) factors. Our own interviews with practitioners are presented in Chapter 4.

3.2 The review process

The SLR procedure we followed[2] is an adaptation of Kitchenham's guidelines which was adapted specifically for SE (Stapić et al., 2012; Wohlin, 2014). After establishing a review protocol, research questions were formulated to steer the reviewing process. These questions, together with the search strings and criteria, helped us narrow down the publication result list into a manageable level, filtering out irrelevant papers.

[2] The data collection of this SLR was carried out by the author of this thesis.

The following research questions were identified:

Chapter Research Questions

1. *Which non-technical skills are identified by educators as important to teach software engineering students?*
2. *How have those skills been successfully taught?*

The ACM Digital Library and IEEE Xplore libraries were used as our main search services as they provide export functionalities, and our institution provides full-text access[3]. As our focus is on software (1) engineering/development (2) non-technical skills (3) in higher education (4), the search string used to gather data is:

[3] These libraries contain publications from both conferences and journals.

```
"software" AND ("engineering" OR "developer" OR
"development") AND [synonym] AND ("education" OR
"educational" OR "teaching" OR "curriculum")
```
[4].

[4] The following synonyms of *non-technical skills* were used in the query: *soft skills*, *21st century skills*, *intangible skills*, *human factors*, *interpersonal skills*, *generic competencies*, *social & emotional intelligence*, *people skills*, *non-cognitive abilities*.

Due to the ambiguity of the synonyms, it is impossible to cover all related publications. Therefore, additional ad-hoc searching via index aggregation services such as Google Scholar was needed to make sure we did not miss any major work. A technique called

"*snowballing*" was adopted to consider publications from reference lists of papers in the review pool (Wohlin, 2014). As part of the quality control, papers were required to contain at least some empirical evidence. Papers written in languages other than English were not taken into consideration. Also, to keep the results relevant for modern SE and to further limit the amount of results, publications older than five years (2014) were not included. However, this date limitation has not been applied while ad-hoc searching.

There is a lot of existing literature on engineering education in general. However, the *software* engineering discipline is unique compared to other engineering disciplines because of the complete absence of a fabrication cost and the increased speed of innovation (Young and Faulk, 2010). This could result in different requirements of non-technical skills for each field. Therefore, papers in this review will not be included without the explicit mention of software.

1962 publications were initially screened by based on their title, keeping 146 results. The next screening phase was based on paper abstracts, keeping 60 results. Papers situating outside higher education were excluded as well. The last screening phase was based on the entire publication content, evaluating quality and applicability. In the end, 26 papers remained to be discussed in Section 3.3.

The complete dataset of all considered publications including extracted data can be found at
`https://people.cs.kuleuven.be/~wouter.groeneveld/slr/`.

3.3 Results & discussion

A LOT OF DIFFERENT approaches towards integrating non-technical skills into the curriculum have been found. These approaches maintain different time frames, ranging from very short projects of one day to extensive capstone projects and internships. Results include diverse research methods, from single case studies to literature reviews (Lenberg et al., 2015). The

combined dataset originates from more than 16 countries world-wide, with Germany (8 papers), Israel (3) and USA (3) on top. The advantage of our systematic review is that this combined data supports stronger evidence on the topic than each individual study.

Some papers have a narrow focus, targeting only a single skill (Holzer, Bendahan, Cardia, and Gillet, 2014; Levy, 2018) or a single teaching aspect (Marques, Ochoa, Bastarrica, and Gutierrez, 2018; Soska, Mottok, and Wolff, 2015; Zheng, Zhang, and Li, 2015). Others are fairly broad, discussing what they call soft skills more generally (Cukierman and Palmieri, 2014; Garousi et al., 2018). We will discuss the results found of each literature review question (`Q1`, `Q2`) individually, concluding with connections between the two.

Q1—Which skills are perceived as important?

Table 3.1 contains a list of extracted non-technical skills, identified as important to teach SE students. Due to the vague definitions of each term, interpretations might overlap. Elaborate descriptions of the found skills were mostly absent in reviewed publications, making it more difficult to generalize or group results.

We have refrained from using our own interpretation of these terms and only marked a term for inclusion in a certain publication if the exact same word appears in the text. Also, note that the absence of a term does not mean that the authors do not deem it as important to teach. It may yet be included implicitly in syllabi programmes.

The reported skills in Sedelmaier and Landes's (2014b) SWEBOS correspond roughly with the results from Table 3.1. SWEBOS uses the following skill groups: *collaboration with others*, *communication*, *structuring one's way of working*, *personal competencies*, *consciousness of problems*, *problem solving* and *further competencies*. It is difficult to assess whether competencies from SWEBOS, such as "*acceptance of responsibilities*", being a combination of role awareness, motivation and leadership. The same applies for "*handling criticism*" or "*working calmly and efficiently under stress*": they show overlap with the skills but cannot be identified with a single one. As a consequence, the

Table 3.1: Identified skills to teach students.

Key	Skill	#papers
S01_Comm	Communication	25
S02_Team	Teamwork/dynamics	25
S03_Refl	Self-reflection	13
S04_Conf	Conflict resolution	13
S05_Mntr	Mentoring	10
S06_Ledr	Leadership	7
S07_Moti	Motivation	6
S08_Role	Role awareness	4
S09_Cult	Cultural Intelligence	4
S10_Crea	Creativity	4
S11_Ethi	Ethics	3
S12_Lifl	Lifelong Learning	3
S13_Empt	Empathy	2

SWEBOS list is not directly visible in Table 3.1.

Communication and **teamwork** were the most commonly identified skills, and have also been the most common identified in industry surveys (Radermacher and Walia, 2013). We did not make the distinction between written and oral communication, such as presentation skills. Holzer et al. (2014) advocate for a separate course devoted to communication while others integrated it more implicitly into the curriculum (Andersson and Logofatu, 2018; Galster, Mitrovic, and Gordon, 2018; Sedelmaier and Landes, 2014a). There is a clear correlation between communication and teamwork: when one skill has been identified in a paper, the other has also been found. Each publication discussing an educational adaptation contained both skills. They go hand in hand: it is only logical that good teamwork requires solid communicative skills. The term "teamwork" also is very common within SE education, as students usually need to finish at least one form of project within a team during the program.

Mentoring and being mentored has been identified as a skill for both students and teachers. Students can act as an "advisor" (mentor) in special programs (Ebentheuer, Kammermann, and Herzog, 2017), or can be mentored by peers or the teaching staff. Most papers left room for mentoring as part of the (capstone) project. Most mentoring happens outside of classrooms, such as the 25% time spent as part of the "Communications & Networks"

course design outlined by Cukierman and Palmieri (2014).

Conflict resolution also appears in conjunction with teamwork and communication. We identified the skill mostly as the classic inter-personal mediation skill when working together on capstone projects (Zheng et al., 2015). However, some papers introduced conflict resolution as an intra-personal skill when developing your own career (Acheson and Rybarczyk, 2016) or thinking about global issues introduced in a communications course (Holzer et al., 2014).

Leadership suggests taking on a leading role in student team projects (Patacsil and Tablatin, 2017; Zheng et al., 2015) or group discussions (Holzer et al., 2014). It can also imply spontaneously taking on the role of mentor when a fellow student is in need of help. There are clear connections between leadership and teamwork: being a good leader will show your ability to work well within a team.

Self-reflection comes in many forms, ranging from general self-improvement (Ebentheuer et al., 2017; Thurner, Schlierkamp, Böttcher, and Zehetmeier, 2016) to specific reflections on the skills learned in the form of surveys (Sedelmaier and Landes, 2014a; Soska et al., 2015) or assessment tools (Marques et al., 2018). The better the student's ability to reflect, the better the ability to absorb other skills. Ebentheuer et al. (2017) reported on a soft skill guidance program that ran successfully for years at their faculty, in which self-reflection plays a central role. It is also important when working with an interdisciplinary group of students (Andersson and Logofatu, 2018), or when thinking about one's future role as a software engineer in society (Acheson and Rybarczyk, 2016). Therefore, self-reflection can be seen as *the* main enabling skill that increases the likelihood of learning anything else. Educators acknowledge its importance:

> *Self-reflection is a crucial enabler for self-improvement in all areas of life.* (*Thurner et al., 2016*)

Motivation as a separate skill denotes the importance of being driven to learn new skills. Students are likely to be more motivated when consistently working together (Tamayo Avila, Van Petegem, Ochoa, and Hernández, 2018). 4 out of 6 occurrences of motivation

also included self-reflection.

Career/Role awareness also requires some self-reflection to see how a software engineer can provide a meaningful role in our modern society (Li, Fagan, and Bourguiba, 2016). Acheson and Rybarczyk (2016) advocate for a deeper understanding of specific strengths in different engineering roles.

Ethics appears in conjunction with role awareness in (Li et al., 2016). Ethics of SE is a topic that usually appears in courses such as "Soft Concepts of Computer Science" introduced by Hazzan and Har-Shai (2013).

Cultural Intelligence/Diversity has been explicitly mentioned in 4 papers. It is a critical skill for future developers as engineering teams can be culturally diverse. This is especially the case with global software development.

Empathy is closely related to cultural diversity and ethics, but there was no overlap found in the usage of these terms. Levy (2018) is the only author to completely focus on empathy in an interdisciplinary course.

Creativity scores surprisingly low at only 4 occurrences. It is mostly related to open assignments in project-oriented learning (Chassidim, Almog, and Mark, 2018; Ebentheuer et al., 2017; Maxim, Brunvand, and Decker, 2017; Soska et al., 2015). We firmly believe that creativity is important to arrive at a good software solution, although hardly any explicit attention is paid to it[5].

[5] We will zoom in on creativity in Part III and Part IV.

Lifelong Learning is the odd one out among the identified skills. As explained in Chapter 2, it describes a set of skills, such as creativity, leadership and problem solving in general. Lifelong learning is an attitude, not an individual skill. However, since it was explicitly mentioned in papers, we decided to include it in the results.

Q2—How are these skills taught?

Table 3.2 contains a list of important aspects found in the included publications to fortify the teaching of various skills identified in `RQ1` (a) and the level at which each aspect can be integrated into the curriculum (b). Both findings are equally important to answer research question two, and will be discussed below, starting with

levels of integration.

Projects (9 appearances) and **Lectures** (13) within a specific course are unsurprisingly the most popular ways to enable learning of these skills. These have always been the classic tools for introducing new goals in the curriculum. Project courses can be converted completely into *service-learning* projects, to contribute to the community and further practice soft skills (Acheson and Rybarczyk, 2016). These projects involve working on real-world problems beyond university boundaries. Service-learning projects also strengthen the bond between community and the university.

Key	Level	#papers
L1_Course	Lectures of a single course	13
L2_Projec	Projects within a course	9
L3_Curric	Throughout entire curriculum	5
L4_Capsto	Capstone projects	4
L5_Intern	Internships	2
L6_Module	Modules within a course	1

(a) Identified levels to integrate skills into the curriculum.

Key	Teaching aspect	#papers
A1_Inter	Interdisciplinarity	7
A2_Group	Group composition	5
A3_Colla	Collaborative Tools	4
A4_Testi	Assessment Tools	4
A5_Activ	Active Learning	3
A6_Video	Video Watching	2
A7_Playf	Playful Learning	2

(b) Identified important aspects of teaching skills.

Table 3.2: Different levels and teaching aspects encountered in the papers that are included in the SLR.

Curriculum-wide incorporation is the ultimate way to integrate non-technical skills into every facet of the SE program, spread across every academic year. "Soft skills must be included in curricula", according to Garousi et al. (2018), while Sedelmaier and Landes (2014b) propose SWEBOS to guide curriculum changes, instead of looking at SWEBOK.

Capstone projects and **Internships** might require more time than a few seminars, but the effort is worth it in the increased amount of skills learned (Zheng et al., 2015). Capstone and internship projects pay off even more if they are real projects developed in-company instead of at the faculty (Bastarrica, Perovich, and Samary, 2017). These initiatives might not lead to completely mastering skills as teamwork and conflict resolution, but to learning their relevance:

> *We designed the course so as its main learning outcome is that students internalize how relevant is having and developing critical soft skills to*

> *succeed in software development projects.* (*Bastarrica et al., 2017*)

Modules within a course, such as interdisciplinary seminars, integrate specific parts in an existing course, without completely redesigning it. For instance, Acheson and Rybarczyk (2016) invite industry experts for seminars in their career development course. This integration is a good alternative, as it:

> *(1) Does not require much additional classroom time and instructor efforts; (2) can be seamlessly integrated into existing course materials; and (3) can start from the student's freshmen year and continues through their undergraduate study in the program.* (*Acheson and Rybarczyk, 2016*)

Interdisciplinarity, as the most important teaching aspect (7 appearances), refers to the mixing of teaching staff and students between faculties, but also between industry and academia. Chatley and Field (2017) introduce industry-relevant content by inviting guest speakers that talk about real-world problems. The communications module of Holzer et al. (2014) made the staff of the Technology faculty work together with the Social and Human Science faculty. Vicente, Tan, and Yu (2018) sent students divided into interdisciplinary teams across programs on a 3-day team-building event to reinforce team spirit. This cross pollination has proven to be more effective to teach empathy, cultural diversity and ethics.

Group composition turns out to be another major factor in enhancing teaching of mentoring, conflict resolution and leadership (Mujkanovic and Bollin, 2016). Research by Tamayo Avila et al. (2018) shows which factors affect cohesion in student teams. Cohesion has been shown to influence motivation, productivity and performance. Teaching staff might put together teams themselves, as was the case with the interdisciplinary teams. Alternatively, surveys may be used to group students with the same primary goal, strengthening some competencies within the group (Thurner et al., 2016).

Collaborative tools have proven to be effective in assisting the skill learning process. Papers reported on the usage of digital tools such as forums and social media to facilitate learning communication (Chan, Chan, Lau, Liang, Huang, Cao, and

Tahernia, 2017; Cukierman and Palmieri, 2014). However, these tools can just as well be simple analog post-it notes when employing agile practices in a project (Chatley and Field, 2017), or when reflecting upon the learned knowledge (Thurner et al., 2016).

Assessment tools have to be developed to evaluate students' abilities during and at the end of a course (Thurner, Bottcher, and Kamper, 2014). Soft skills are, compared to hard skills, difficult to pin down in terms of grading[6]. The speed of evaluation matters: employing a "*fast feedback cycle*" allows students to practice their reflection skill more often (Chatley and Field, 2017).

[6] An assessment tool for evaluating creativity is presented in Chapter 11.

Active Learning and **Video Watching** help foster further learning of soft skills. Galster et al. (2018) opted for what they call "*Active Video Watching*", integrating interactive activities into videos to reduce the resource costs involved in teaching. Videos are also used as supportive material together with classes (Holzer et al., 2014). Especially for soft skills, active engagement with others is required to construct mental models. This interaction itself again requires the application of soft skills (Hazzan and Har-Shai, 2013).

Playful learning has been used in the form of gamification (Maxim et al., 2017) and experimental investigations to discover unknown content (Soska et al., 2015). Learning through play with supportive guidance creates more space to discover, improvise and challenge, directly influencing creativity and motivational skills. Playful learning however still is in an early stage of adoption in SE education, according to our 2 included occurrences.

Relationships between results

Figure 3.1 visualizes the relationship between identified skills and topics relevant for teaching them. Teaching aspects and levels of integration from Table 3.2 have been combined, totaling 13 topics, to put emphasis on skills.

The following interesting connections have been discovered by interpreting the visual links between skill and teaching topic, or the striking absence of any link where we would expect one to be[7].

[7] These findings are easier to deduce from the interactive version of this diagram which is available at `https://people.cs.kuleuven.be/~wouter.groeneveld/slr/`.

General popular combinations: self-reflection and conflict resolution each appear in 13 out of 26 papers (50%) included in this study. Each time either is mentioned, all identified teaching

aspects are also mentioned. The same effect holds for the two most common appearances, communication and teamwork.

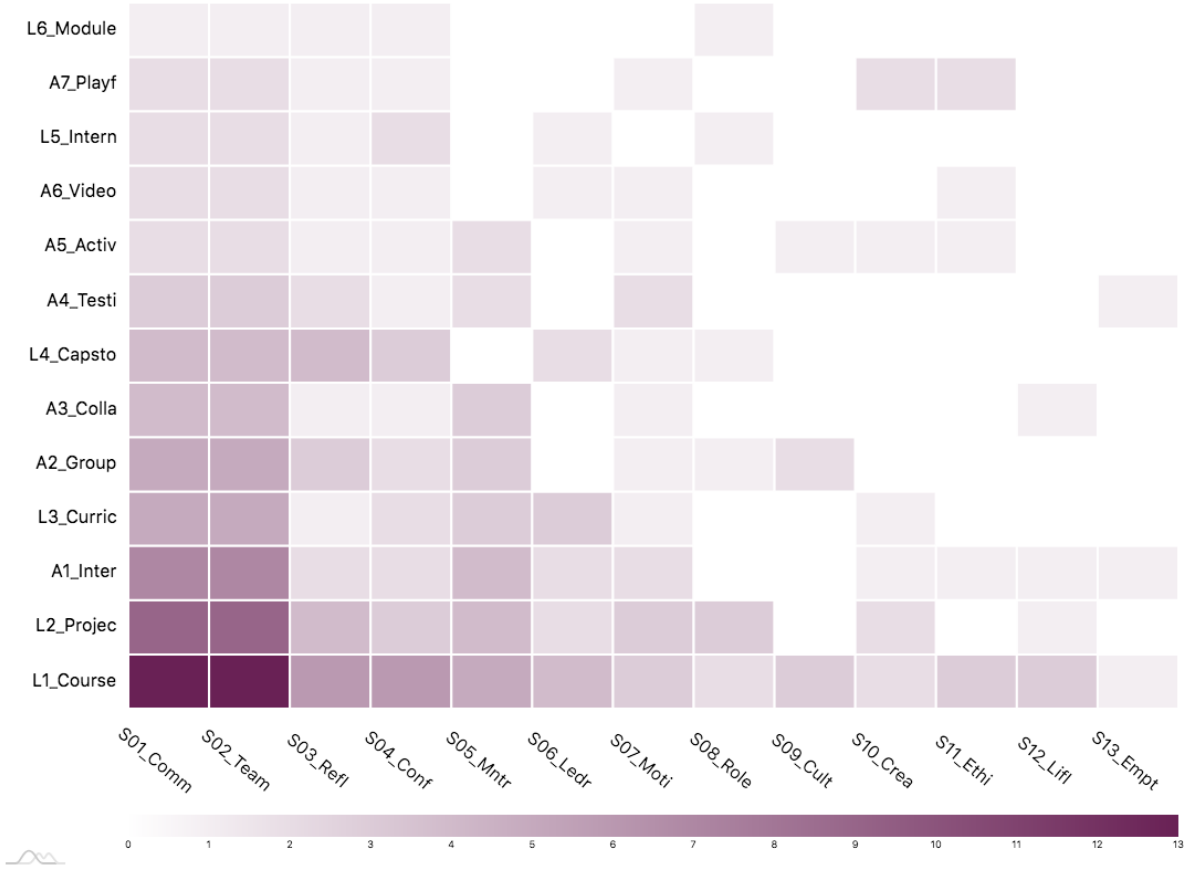


Figure 3.1: An overview of the relationships between skill (x-axis), aspect and level (y-axis), outlined as a heatmap. Keys can be translated into corresponding values via Table 3.1 and Table 3.2. Interactive visualizations of this and are available at `https://people.cs.kuleuven.be/~wouter.groeneveld/slr/`.

These four abilities are the top studied skills in publications. Motivation is the next highly linked skill, missing only internships and modules in existing courses. It surprises us that capstone projects are barely linked with motivation (1 paper), as one would expect that motivation is needed to fulfill a such a project, and that these projects are more motivational than solitary, unlinked exercises.

The most conventional and top studied method, lecturing as part of a single course, is identified with each found skill. Another

interesting finding is despite the low appearances of some aspects, these show a relatively high cohesion towards learning multiple skills.

***Collaborative Tooling* and *Test Assessment* show promise**: While collaborative tools assist the learning of communication skills, it is not yet used to facilitate the teaching of ethics, empathy and cultural diversity. Forums and social media could also be deployed for these abilities. Testing students' behavior on non-technical abilities also shows encouraging results. We believe this can be further extended by involving role awareness, ethics and leadership. Creativity is much more difficult to assess using tests.

***Internships* and *Capstone Projects* require more attention**: Internships are only connected to leadership and role awareness, besides the top studied skills. Capstone projects should also better support motivation. These levels of integration showcase a surprisingly low number of skills to be taught, especially considering the outcry to bring industry and academia closer together. Strangely enough, there is only a very weak link between leadership and internships, and none at all between mentoring and capstone projects.

***Interdisciplinarity* facilitates teaching beyond the Technical**: While interdisciplinary teaching aspects are only employed in 7 out of 26 papers (27%), they do cover all skills except role awareness and cultural diversity: 11 out of 13 skills (85%). Employing an interfaculty team, or even an interdisciplinary team across industry and academia, is strongly recommended (Holzer et al., 2014). Interdisciplinarity can be used together with other work forms.

***Lifelong learning* is perceived as a secondary goal**: The low number of appearances of lifelong learning (3), combined with the low number of coupled teaching aspects and levels (4 out of 13, 30%), leads us to conclude that teaching students the importance of continuous training by itself is considered only a secondary or implicit goal, being (semi-)automatically induced by other skills. We believe it should instead be given the greatest attention, especially in an ever-changing world like software engineering. Figure 3.1 confirms the relationship between lifelong learning and

self-motivation or self-reflection.

***Creativity* is absent in bigger project development**: creativity appears in 4 out of 26 papers (15%), together with the following work forms, as indicated by the highlighted lines of Figure 3.2: course design (`L1_Course`), project design (`L2_Projec`), interdisciplinarity (`A1_Inter`), program design (`L3_Curric`), active learning (`A5_Activ`) and playful learning (`A7_Playf`). While it is related to a reasonable number of teaching aspects (6 out of 13, 46%), it is not explicitly found when students embark on bigger projects such as capstone projects and internships. The papers included in our result do not focus explicitly on creativity alone.

***Lectures* might not be ideal to induce non-technical skills**: It is interesting to note that lectures, appearing 13 times (50%) and connecting with all skills, also seem to be preferred to induce more practical skills. Perhaps this is simply because it is a well-established method to teach theoretical knowledge. One could ask whether this is the most effective way to engage students.

3.4 Limitations

A POSSIBLE THREAT TO the correctness of our results is that the list of non-technical skills we identified in Table 3.1, or the teaching aspects and levels in Table 3.2, may be incomplete. Even though we recognize this possibility, we consider it unlikely, given the used SLR methodology and additional snowballing technique we adhered to, which reduces the risk of making these errors.

Limited visibility of publications may have led us to exclude certain important work. Most papers focus on soft skills in general, but some are more devoted towards certain individual skills. This will influence the visualization of the relationships between skill and aspect. Also, our SLR query did not yield negative results (e.g. papers claiming that capstone projects do *not* contribute to creativity). It seems that more positive than negative results tend to get published, possibly leading to an over- or underestimation of certain terms found in our results.

Since the data extraction process is a manual process, we cannot

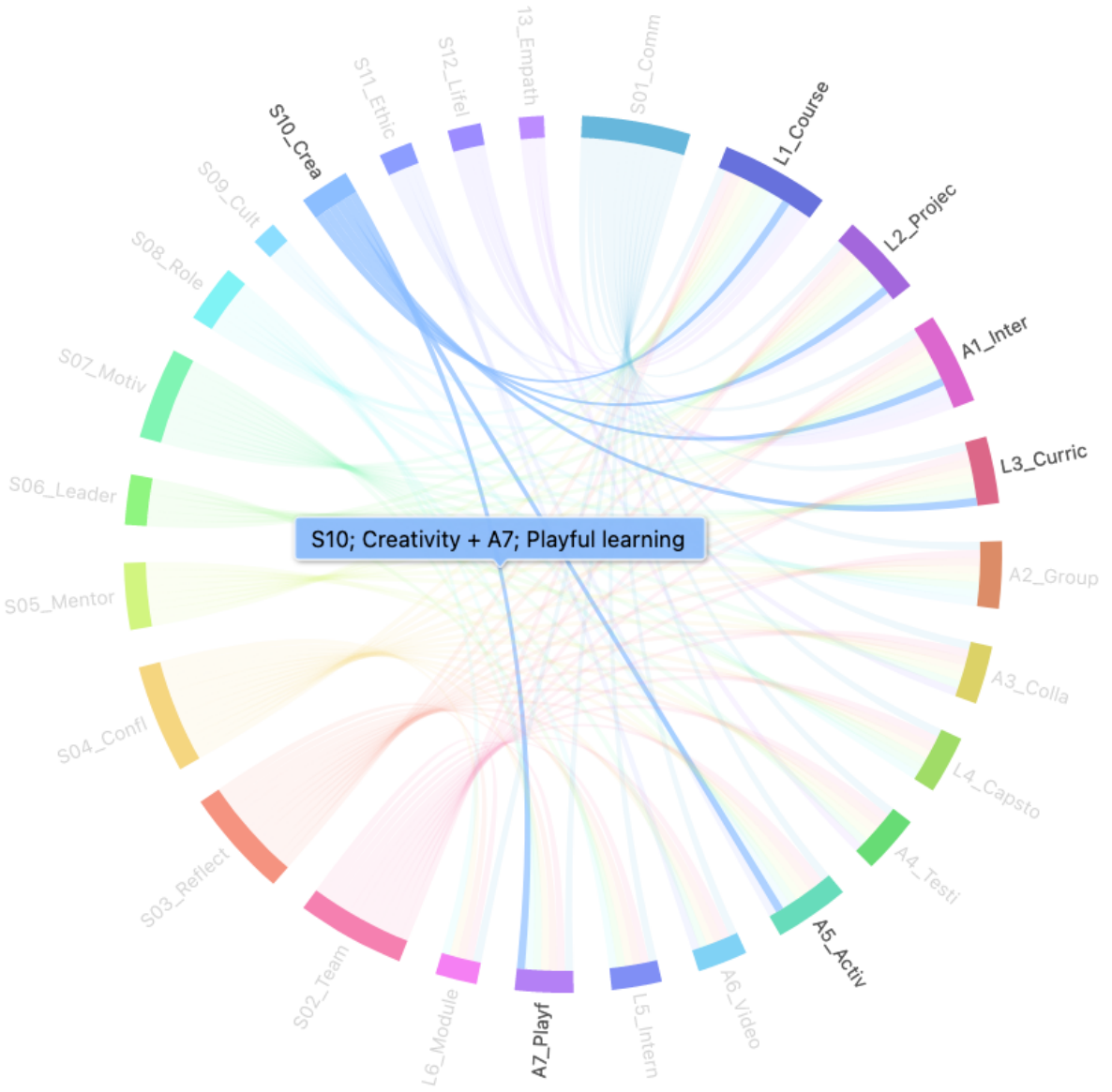


Figure 3.2: A chord diagram highlighting the link between creativity as a skill and encountered educational work forms. Consult the interactive version for a better experience.

guarantee that some papers, skills or aspects, have not mistakenly been excluded or missed. Therefore, we discussed our results as a whole without drawing conclusions based on a single paper.

3.5 Conclusion

The results of our SLR based on 26 papers identify which non-technical skills are perceived as important to educators, and how these are currently being taught. This overview presented some of the most common aspects to facilitate the teaching of these skills. We discussed each skill, aspect, and integration level individually to provide some context, highlighting the most and least common occurrences. By looking at the relationship between skill and aspect with the help of Figure 3.1 and Figure 3.2, we discovered popular combinations and interesting trends in software engineering education.

It is clear to us that the result of collaborating across academia and industry has a major impact on teaching non-technical skills. The first steps have already been taken to successfully blend practical psychology and philosophy with SE, but there is still room for improvement, both in depth and in breath.

Based on the results and conclusions of this chapter, we reckon that the following steps should be taken to strengthen the current SE curricula. First, the program should focus more on interdisciplinary teaching, not only by inviting lecturers from other faculties, but also from outside the university[8]. Next, non-technical skills should be examined in more detail in combination with external internships and capstone projects. Lastly, skills such as creativity and a strong emphasis on lifelong learning should be induced in all available courses, including technical ones.

[8] Our own interdisciplinary intervention will be discussed later, in Chapter 14.

The findings of this chapter serve as a foundation to further investigate how to integrate the learning of non-technical skills into the curricula. For instance, to further investigate certain skills or teaching methods based on the greatest common denominator, in Chapter 6, we will compare the results of this chapter with findings from the industry surveys presented in the following chapter.

4 Non-technical Industry Requirements

Publication details

This chapter has been published in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* as *Non-cognitive abilities of exceptional software engineers: a Delphi study* (Groeneveld et al., 2020b).
Its main contributions are uncovering which skills software developers deem as needed to excel in the industry and comparing opinions of technical experts, business experts, and academics.

We now have an idea of how non-technical SE skills are currently being taught in academia. The SLR study from Chapter 3 revealed that multiple attempts to categorise skills have been made, but so far little international studies have been performed that identify skills by interviewing experts from both the industrial and academic world.

In this chapter, we present the results of a Delphi study, where 36 experts from 11 different countries were invited to help identify which non-technical skills[1] are needed to excel in the SE industry. The 55 uncovered and ranked skills as 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,

[1] In the published paper, the term *non-cognitive abilities* was first used, before asking interviewees for alternatives. See Chapter 2 for a discussion on the terminology.

solution-oriented thinking, etc.), and *personal skills* (curiosity, being open to ideas, etc.), of which a comparison has been made between opinions of technical experts, business experts, and academics.

The Delphi study sought to answer the following questions:

Chapter Research Questions

1. *Which non-technical skills are needed for a developer, to excel in the software engineering industry?*
2. *What is the relative importance of these skills?*
3. *Does the opinion of industry experts differ from academics, and if so, in what way?*

We first discuss similar works before continuing to unfold the Delphi methodology and presenting the results of the study.

4.1 Related Work

What makes a great software engineer? is the central question Li et al. (2015) answer by interviewing engineers at Microsoft. Li identified a set of 53 general attributes of great engineers, categorized into *personal characteristics* (improving, passionate, open-minded, etc.), *decision making* (knowledge-oriented), *teammates* (mentoring, asking for help, honest, etc.), and *software product* (elegant, creative, evolving, etc.). Even if the interviewed engineers come from different divisions, the identified attributes still originate from only one company. While the diverse set of attributes certainly help to understand what affects the success of software projects, the lack of prioritization renders it difficult to translate this list into concrete recommendations for education.

A lot of similar investigations exist in literature, but few seem to primarily target non-technical skills. Some researchers look for industrial requirements by analyzing job ads, risking the inclusion of buzz-words and absence of implicitly required skills (Stevens and Norman, 2016). Ad content analysis and surveys are popular tools used to research trends and identify (IT) skills. However,

Delphi studies usually offer more depth in terms of identification of a single concept, such as critical skills for managing IT projects (Keil, Lee, and Deng, 2013), or soft competencies in requirements engineering (Holtkamp, Jokinen, and Pawlowski, 2015). Our aim with this study is the same, focusing instead on modern software development.

Another Delphi study was conducted by Wynekoop and Walz (2000) to identify the traits of top performing software developers. This study revealed the most important high-level traits to be geared towards cognitivism. Of course, within a period of 20 years, the software development world has evolved considerably, as the results of this chapter indicate.

So far, we have not found a study that includes both the opinion of people from the industry and people from the academic world. It will be interesting to see what both parties have to say. As is the case with most related studies, high-level discovered traits such as *good interpersonal skills*, *team oriented*, *leadership skills*, and *technical proficiency* lack depth, serving as a whole category rather than an individual trait. To mitigate this shortcoming, we made sure to break down general skills into multiple specific low-level ones, as revealed in Section 4.3.

4.2 The Delphi process

To be able to answer our research questions without merely relying on analysis of existing data, we needed the help from industry veterans. Since simply identifying skills would not sufficiently answer our questions, we opted for the Delphi approach, of which we have applied the most conventional implementation, collecting opinions of different groups called *panels*. The Delphi technique is an effective tool to reach a group consensus on a given topic, where the opinion of experts is used for decision making.

As stated by Okoli and Pawlowski (2004), expert selection is one of the most important aspects in a Delphi study. We adhered to their proposed participant selection model by identifying relevant

stakeholders, carefully reviewing their expertise based on information at hand, such as blogs, CVs, social media, and papers, categorizing the identified experts into separate stakeholder panels, and inviting experts until a pre-set limit of 10 to 18 members per panel is reached.

Participating in a Delphi study requires a hefty commitment, since there are several rounds to go through, with the first open-ended round being the most cognitive demanding. Because of that, the availability of information regarding the process and a correct formulation of questions were given special attention. Potential panel members were contacted and asked to participate to reduce drop-out rates during the study.

We invited experts from three different fields, as visible in Table 4.1: (T) *technical experts* (software developers, software architects, technical coaches, etc), (B) *business experts* (development and HR managers, agile coaches, analysts, etc), and (A) *academics*. The first two groups consist of people in daily close contact with SE, while the last group contains researchers and/or educators with experience in the same field.

Table 4.1: Panel groups, response rates, and mean duration in minutes for each phase.

Round	**Dur. (m)**	**Panel T**	**Panel B**	**Panel A**
Invites		35	31	80
Members		14 (40%)	12 (39%)	10 (12%)
#1	35	11 (79%)	11 (92%)	10 (100%)
#2	6	11 (79%)	8 (67%)	7 (70%)
#3	6	9 (64%)	10 (83%)	9 (90%)
#4.1	10	9 (64%)	9 (75%)	8 (80%)
#4.2	6	9 (64%)	8 (67%)	9 (90%)

Our aim at answering the research questions was to deliberately not limit ourselves to collecting responses from domestic software engineers, but to also include opinions of as many widely recognized experts as possible from well-known international companies. Hence the choice of distributing the survey through email. This choice comes with the unfortunate downside of initial low response rates combined with a possible drop-out between rounds.

For this study, the Qualtrics web-based survey system was used.

Information was gathered as follows: starting with an open-ended questionnaire (1), after data processing and grouping, we ask participants to verify and potentially revise their answers (2), continuing with the selection of the top x most important skills (3), to finish the survey with a series of ranking iterations (4), until a consensus is reached or two iterations have passed. We will briefly discuss each step below, as shown in Figure 4.1.

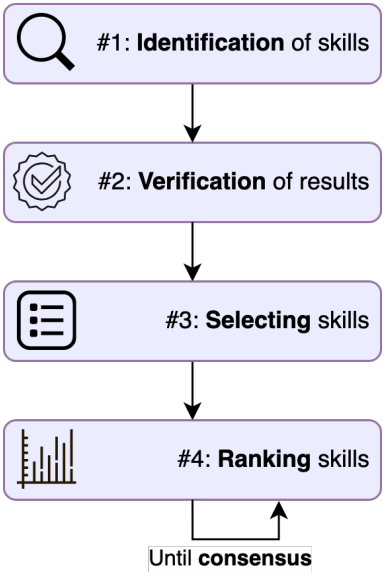


Figure 4.1: The Delphi survey structure.

1. Identification of skills: The first questionnaire is the most crucial one, designed as an open-ended digital brainstorming session. Participants are asked to answer our first question by listing at least *five skills* they think are important. Because we expected this to be answered with typical high-level keywords such as "communication", "teamwork", and "collaboration", we asked two more questions. First, respondents were required to include a brief explanation (2 to 3 sentences) of *why* these are important to them. Next, they had to imagine a developer (possibly themselves) working with a colleague on a problem, specifying *what behavior* they observe that shows this colleague has the qualities answered in the first question. This extra data made it possible for us to move from a general keyword to a specific skill. In order to avoid confusion regarding the terms used in the questions, we provided a definition[2].

2. Verification of results: To ensure the correctness of coding the open responses from round 1, we recruited the help of a seasoned software developer[3], who was familiar with the problem at hand, but did not participate in the survey. He analyzed two thirds of the anonymized raw data, creating his own set of categories. After comparing and correcting the results, a final run through all the responses resulted in the finalized list of identified skills. After this coding step, the next questionnaire asks respondents to identify their answers, and if needed, provide further suggestions. This step in the Delphi process verifies the correctness of our coding system, and gives panel members a chance to add to their previous answer. When discussing the results, duplicates were eliminated, so that the original wording of the respondents was retained as much as possible.

3. Selecting most important skills: Now that the complete list

[2] As discussed in Chapter 2, we ultimately switched terminology. For the sake of completeness, the old definitions as provided in this study are included in Section 2.2.

[3] The second author of Groeneveld et al. (2020b).

of skills has been identified, we asked participants to select *at least 10 factors* they consider to be the most important skills for a developer to excel in the SE industry. In this round, categories have been removed, and the order of all choices has been fully randomized. It is important to notice that from round three and up, the list of skills will be narrowed down and ranked from within each panel, while the first two rounds were conducted jointly by all panels.

4. Ranking skills: To reduce the ordered skill set from round 3 into a manageable size for each panel, only selected skills by at least one member of this panel will remain for respondents to rank by importance, using a drag-and-drop interface. The last round has to be repeated until a general consensus is reached, measured by calculating *Kendall's W* >= `0.7` (`1` = complete agreement, `0` = no agreement) (Okoli and Pawlowski, 2004). If *W* does not rise to the desired level within two iterations, the process will stop anyway, as to not overburden panel members with questionnaires. The end result of the Delphi study nets us three separate lists of ranked and important skills, identified by each group of stakeholders.

The results in between rounds were not fed back to the participants, except for the data needed to compose the next round. Everyone received a summary of their own answer after each round. Panel members did not know each other, but their identity was visible for the researchers. Data from drop-outs was not excluded from subsequent rounds, and no additional people were invited once the study started.

4.3 Results & discussion

Panel members completed the survey from 11 different countries world-wide, with 17 members from Belgium. Members came from 12 different companies and 9 different learning institutions, including our own university. 146 people have been invited to take part, with 36 positive responses, leading to an invite response rate of 25%, with 39% from the industry and merely 12% from academia, of which 89% participants successfully completed

the first round.

Several well-known authorities in the field of enterprise software development took part, as well as (co-)authors of praised books. We made sure not to invite more than four experts from the same company or university, reducing the risk of creating homogeneous groups. Companies involved differed in size, from small to internationally renowned. Thus, we believe panel members to have accumulated sufficient experience to be able to contribute to this research.

Tables 4.2, 4.3, 4.4, and 4.5 collectively contain the full list of 55 identified skill, including a final ranking for each expert panel: technical experts (T), business experts (B), and academics (A). The tables[4] are sorted by occurrence in round 1. Occurrences per panel from round 3 are indicated in gray. Figure 4.2 visualizes the ranked skills present in the tables.

[4] A unified and filterable version of all table data is available at https://people.cs.kuleuven.be/~wouter.groeneveld/delphi/.

Our aim was to code the skills as detailed as possible (e.g. *adjust your communication to the target audience*, *visually communicate ideas*, *speak well in front of an audience*) instead of listing high-level skills (e.g. *communication*, *presentation skills*, *leadership*). Skill categories closely related with cognition (e.g. *intelligence*, *critical* and *analytical thinking*) were also split up, while too technical answers were omitted from subsequent rounds, as these are not relevant to answer the research questions.

There was very little consensus within each panel during the first ranking round: W was calculated at `0.16`, `0.12`, and `0.16` for panel T, B, and A, respectively. After rearranging skills according to the new mean ranks and providing a summarized motivation as feedback, the second ranking round reached a much higher agreement: W reached `0.74`, `0.80`, and `0.58`, proving to be sufficient to end the survey.

Some skills are closely related (e.g. *sharing knowledge* and *coaching others*), but still sufficiently dissimilar to justify separation. However, some participants showed difficulties understanding these subtle contrasts while prioritizing and ranking them in round 3 and 4, contributing to a low consensus rate. We felt that digital surveys did not offer enough room for discussion between panel members to clear out any disagreements on precisely defining

Communicative Skills (11). The ability to...	**Occ. #**	Occ. round 3 - **Final rank** **T**	**B**	**A**
adjust your communication to the target audience	11	4 16th	/	5 12th
actively listen	9	6 10th	5 10th	4 6th
understand and engage with the people involved during development	8	/	/	6 1st
clearly express intentions to avoid misunderstandings (including in code)	8	/	/	/
show respect by exercising patience	6	/	/	/
visually communicate ideas/problems (e.g. drawing on a whiteboard)	4	5 13th	4 13th	/
empathize with others by understanding others' viewpoints	4	4 7th	/	4 4th
be aware of, and to a certain extend control, own and others' emotions	2	/	/	/
have the courage to raise concerns	2	4 15th	/	/
demonstrate maturity by admitting when you're wrong	2	/	/	/
speak well in front of an audience	1	/	/	/

Table 4.2: The 11 identified communicative skills.

Collaborative skills (16). The ability to...	**Occ. #**	Occ. round 3 - **Final rank** **T**	**B**	**A**
collaborate with others to achieve a shared goal	10	4 6th	5 1st	7 2nd
learn from each other by sharing/evangelizing/teaching/... knowledge	9	/	6 6th	/
be accountable by showing, taking, and sharing responsibility	8	4 12th	5 4th	/
give and receive feedback	8	4 17th	6 9th	4 8th
(pro-actively) ask for help when needed	6	/	5 12th	4 13th
(pro-actively) help others when they encounter a problem	5	/	/	/
negotiate a solution that resolves tensions between stakeholders	4	/	5 14th	/
show and build a sense of belonging to team- and companywide values	3	/	/	/
mentor/coach/train others by providing guidance	3	4 18th	/	4 15th
promote well-being by contributing to team spirit	2	/	/	/
maintain a positive attitude that inspires others	2	/	4 7th	/
practice discipline by acting according to established team rules	2	/	/	/
refrain from frequently interrupting others	2	/	/	/
work autonomously within a group	2	/	/	/
resolve conflicts by taking advantage of new perspectives	2	/	/	/
accept diversity in the workplace	1	/	/	/

Table 4.3: The 16 identified collaborative skills.

Problem-Solving Skills (13). The ability to...	**Occ. #**	Occ. round 3 - **Final rank** **T**	**B**	**A**
show perseverance by not giving up easily and keep on trying again	11	4 11th	/	/
be creative by approaching a problem from different angles	10	7 3rd	4 8th	4 9th
show attention to detail	4	5 9th	/	/
focus on complex problems for longer, uninterrupted periods	3	/	/	/
systematically verify assumptions and validate results	3	5 1st	/	5 5th
reframe problems by observing the bigger picture	3	7 8th	/	/
draw on multiple sources and domains for ideas	2	/	/	/
create a clean structure leading to predictability and reliability	2	/	/	/
like the challenge of a difficult problem	1	/	/	/
think in a solution-oriented way by favoring outcome over ego	1	4 14th	7 2nd	/
focus on efficiency and re-use instead of reinventing the wheel	1	/	/	/
measure (own/others/project) progress and improvement	1	/	/	/
know which actions to take under which circumstances	1	/	/	/

Table 4.4: The 13 identified problem sovling skills.

Personal Skills (15). The ability to...	**Occ. #**	Occ. round 3 - **Final rank** **T**	**B**	**A**
be curious by wondering why and how something works	10	7 2nd	/	5 10th
devote oneself to continuous learning	8	8 4th	7 3rd	5 14th
be open to the ideas of others	7	7 5th	4 5th	6 3rd
be a fast learner to keep up with the pace of new technologies	7	/	/	4 16th
be passionate in what you do	5	/	/	/
criticize oneself and others objectively and fairly	4	/	/	4 11th
be flexible by quickly adapting to changes during development	3	/	/	5 7th
be consistent by doing as you say and saying as you do	2	/	/	/
set goals and stick to them	2	/	/	/
be adventurous by willing to take risks	1	/	/	/
carefully choose which new trends to follow and which to ignore	1	/	/	/
be stress-resilient	1	/	/	/
to cope with and learn from failure	1	/	4 11th	/
develop a sense of pride in your work	1	/	/	/
leverage self-knowledge to continuously improve oneself	1	/	/	/

Table 4.5: The 15 identified personal skills.

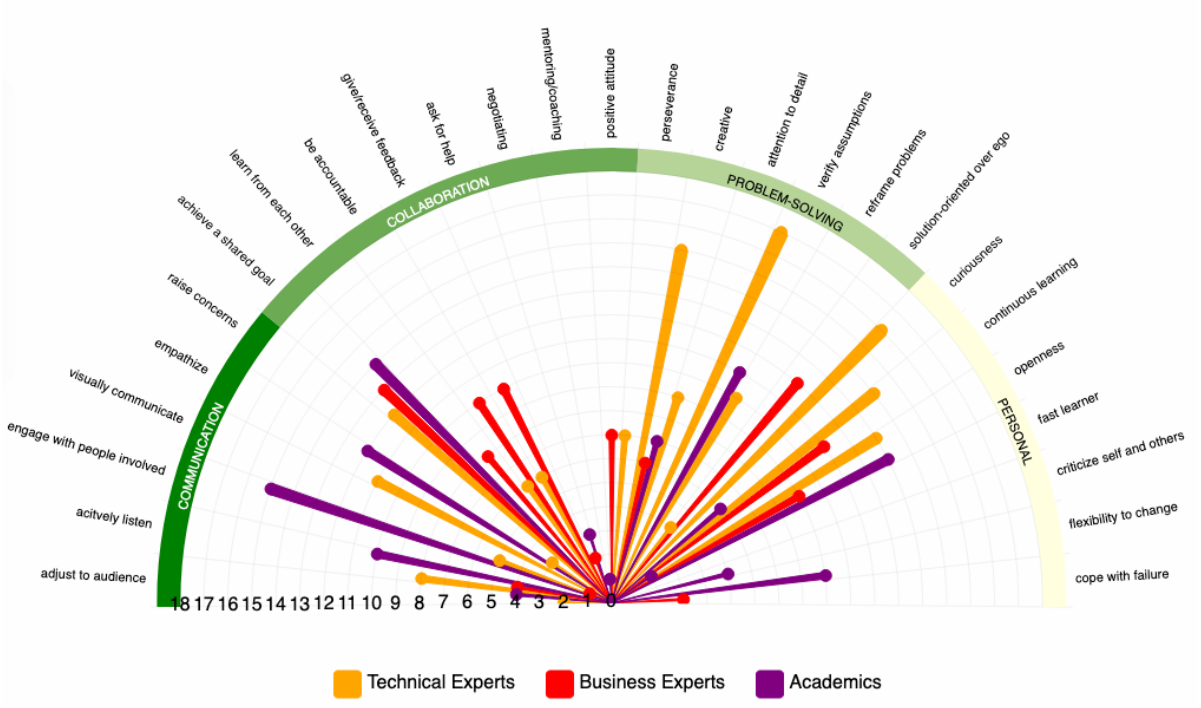


Figure 4.2: An overview of the ranked skills from the Delphi study. An interactive version is available at https://people.cs.kuleuven.be/~wouter.groeneveld/delphi/.

these terms, despite our attempts to answer questions and disseminate information between rounds.

In general, we noticed a clear pattern in the answers, that is reflected in two groups of two categories: communicative skills (1) & collaborative skills (2), and problem-solving skills (3) & personal skills (4). As one participant rightfully observes:

> *To me, software engineering is a balancing act between being able to communicate with other people and solving problems.*

The first group focuses on getting along, while the last two categories contain problem-solving skills and engineering inventiveness. We will briefly discuss each category, highlighting interesting findings, continuing with a look at the degree of consensus for each expert panel.

Communicative Skills

As mentioned in Chapter 3, terms such as "communication" and "teamwork" are most common when investigating skills to teach students. It is therefore no surprise that skills such as *understanding and engaging with people involved*, *adjusting communication to the target audience*, and *actively listening* are reflected in the results. It is interesting to note that technical experts selected 5 out of 11 communicative skills to be of great importance, while business experts only selected 2, and academics 4. *Having the courage to raise concerns* was only picked by panel T and *understanding and engaging with the people involved* only by panel A. Communication is also seen as a way to structure and guide the flow of information within a group:

> *Teams usually have to find a way to communicate, to try out several ideas. Communication also requires structure: a common system to make notes and have weekly/daily meetings to discuss these notes and decide how to continue.*

Frequently mentioned skills during the first round such as *clearly expressing intentions to avoid misunderstandings* (8 occurrences) and *showing respect by being patient* (6) were not deemed important enough to make it to the final rankings.

Collaborative Skills

Being a good communicator is not enough: enterprise software development also requires working together, *in close collaboration to achieve a shared goal*, the most popular skill in this category. It is self-evident that communicative skills also work in favor of collaboration. Therefore, some items are interchangeable between this category and the first.

Business experts especially seem to appreciate these skills, deeming 7 out of 16 of them as important in round 3, with *learning from each other by sharing/evangelizing/teaching knowledge*, *negotiating*, and *maintaining a positive attitude* being skills not picked by the other two panels. However, *mentoring/coaching/training others* was indeed picked, and might also be interpreted as 'sharing knowledge'. Skills that promote *belonging* have been highlighted by

several respondents, reporting that software development is a team sport nowadays:

> *The ability to function in a team, to help with the norming, forming and performing of a team is crucial. In the end it's the team who is committed to the success of the software, so we need developers who are committed to the team.*

The same holds true for *negotiating*:

> *Software is usually written for non-technical customers and stakeholders, for commercial reasons. There is always a tension between technical constraints, commercial needs, the needs and desires of different stakeholders etc. Successful software development involves negotiating a solution that best resolves those tensions.*

However, compared to other items in the result set, these barely made it to the final ranking. *Asking for help when needed* is seen as more important than *refraining from interrupting others*, and *working autonomously* again was given less attention than heavy team-oriented skills.

Problem-Solving Skills

Besides getting along, a second central theme to the skill set of a good software developer is the ability to identify and effectively solve problems. Sedelmaier and Landes's (2014b) SWEBOS splits this category into *consciousness of problems* and *competence to solve problems*. Response data also shows that being able to identify a problem is something different than being able to actually solve it. It comes to no surprise that technical experts marked 6 out of 13 skills from this category as important. What did surprise us was the low amount of these skills picked up by other panels.

Showing perseverance, the most popular skill (11 occurrences), was only chosen by panel T, as did *attention to detail* (4), and *reframing problems by observing the bigger picture* (3). We feel that the ability to *focus on complex problems for longer, uninterrupted hours*, although mentioned (3), is underrated, as this kind of "deep work" or "flow" has been proven to aid in problem solving (Newport, 2016). When it comes to software development, "*the devil is in the details*", as one respondent said:

When telling a computer what to do, the devil is in the details. I have never known a good programmer who doesn't pay careful attention to the details of the problem, the code, and the process.

Many skills, including *creativity*, can be amplified in the presence of other skills. They are connected in a systemic way that is difficult to identify in a Delphi study, apart from analyzing open ended question responses, such as this one:

The ability to think of new and creative ways to do things is essential for any kind of problem solving. but it is particularly important in software where complexity requires good design. Creativity includes curiosity, questioning, exploring multiple paths, and converging on a design.

Personal Skills

The last category represents characteristic attributes of individuals, as a combination between willingness and the ability to use a skill is required to succeed. These dispositions include *curiosity*, *being open to ideas of others*, *self-criticism*, and intrinsic motivation such as *being passionate*, which are all consistently reported in critical thinking research (Facione, 2000). Some items from the previous category can also be placed within the context of personal attributes.

For academics, this category is the most important one, with 6 out of 16 skills that made it to the final ranking, while other panels only include 3. While *being a fast learner to keep up with the pace of new technologies* (7 occurrences), *criticize oneself and others* (4), and *being flexible by quickly adapting to changes* (3) are popular choices, they failed to stay on target except in panel A. *Coping with and learning from failure* is an ability picked only by business experts. Developers do acknowledge the need for resilience, as one panel member stated:

Things will go wrong, you will make mistakes, you will misunderstand requirements or technology. You need to be able to not just cope with, but learn from that failure.

This is also called "*creating a safe haven*" by Li et al. (2015), where engineers can learn and improve from mistakes without negative consequences.

Expert panel opinions

Perhaps unsurprisingly, industry and academic experts reached a significant difference in opinion, as also reported Eskandari, Sala-Diakanda, Furterer, Rabelo, Crumpton-Young, and Williams (2007) in a Delphi study to suggest engineering curriculum enhancements. Figure 4.3 visualizes these differences by comparing the normalized sum of ranked skills as weights for each category.

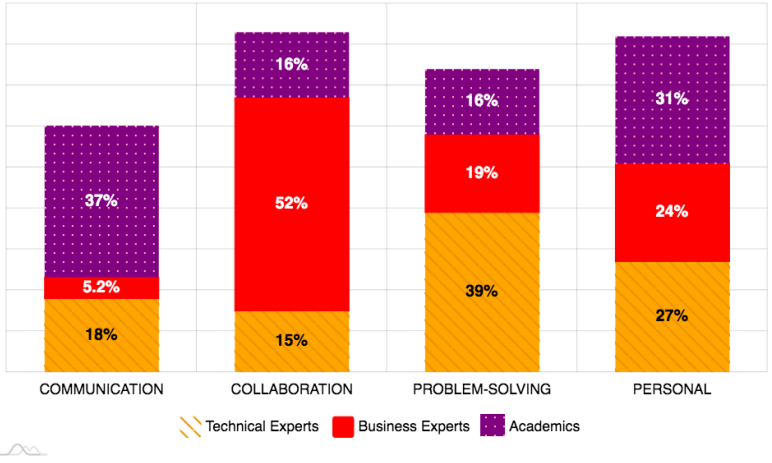


Figure 4.3: A visual comparison between expert panel opinions, divided in the four skill categories. Orange-striped: Technical Experts. Red-plain: Business Experts. Purple-dotted: Academics.

Technical experts (panel T) claim *systematically verifying assumptions* (#1) is one of the most important skills to be successful, together with *curiousness* (#2) and *creativity* (#3). Without *continuously improving* (#4) and learning, any necessary non-cognitive skill becomes unsustainable. Problem-solving skills and personal skills matter most.

Business experts (panel B) hold *collaborating to achieve a shared goal* (#1) in high regard, next to *thinking in a solution-oriented way* (#2). After all, the most important deliverable for a software developer is working software, amplified by shared success that may involve some personal compromises (Li et al., 2015). *Continuously improving* (#3) comes in third. There are some similarities between business and technical experts. A good balance between problem solving, personal skills, and

collaborative skills is required, with a heavy emphasis on collaboration, as visible in Figure 4.3.

Academics (panel A) consider *understanding and engaging with people involved* (#1) to be an important aspect, together with *collaborating to achieve a shared goal* (#2) and *being open to the ideas of others* (#3). They conclude that personal skills and communication are of the utmost importance. The difference in opinion about communicative skills between academics (37%) and business experts (5%) is striking.

A possible explanation for the heavy emphasis on problem solving for technical experts could be the tangible and immediate usability, while more abstract personal characteristics that are preferred by academics help with the norming and forming of those practical skills. However, the industry seems to expect ready-to-use practical problem-solving skills (Bailey, 2014). This may depend on your professional experience, as one respondent points out:

> *I found it very difficult to rank the skills by importance. Relative importance can depend on where you are on your career.*

This is another reason why reaching a certain degree of consensus within each group was not easily achievable.

4.4 Threats to validity

By carefully following the steps explained in section 3, and by comparing our data with existing research, we are convinced that our results can answer `Q1` and `Q3` safely. However, since a Delphi study never reaches any statistical relevance, the answer to `Q2` regarding relative importance of each skill might vary, depending on the formation of the panel. Also, since the results are highly contextual, ranking them might not always be very relevant.

As mentioned before, depending on where you are in your career, certain skills might be more important to you than others. Also, depending on the interpretation of the questions or the skills themselves, answers might vary. It is very difficult to facilitate a heated debate using only a digital survey.

This could be mitigated by performing additional focus groups using the results from this study as input, or by repeating round 3/4 in Figure 4.3 with a new set of experts, until the outcome remains steady. Regional and cultural differences might further complicate things. Generalization was, however, never the goal for this research. We believe that our systematic procedure yielded interesting and relevant findings.

4.5 Conclusion

The first overarching research question of this dissertation, as presented in Chapter 1, was:

PhD Research Question 1

What exactly are the non-technical industrial requirements of modern software engineers?

The 36 panel members of the Delphi study presented in this chapter agreed upon important non-technical skills needed for a developer to excel in the SE industry. This, combined with the results of the literature review of Chapter 3, thereby successfully answered our first question, and closes out Part I.

Furthermore, by visualizing and comparing results, we have uncovered interesting differences of opinion between panels. These findings will hopefully help to closing the gap between industry and higher education. Future work might dig deeper into understanding these dissimilarities.

The legitimate remark regarding your personal career stance raises the question of which skills are absolutely essential for a starting developer, and which skills will be developed as your experience grows. The skill categories and voting trends visible in the tables are ultimately more interesting for further investigation than the level of agreement within the ranking rounds.

It is important to ask ourselves which skills from the list should be taught to students, and which skills they will develop themselves as practitioners. When looking at the results of Chapter 3, communication and teamwork are well-researched and taught in higher education, while certain skills identified as

important such as creativity and empathy seem to be severely underrepresented in SE curricula.

To further reconcile industry and academia, in Part II, we will zoom in on underexposed skills that were considered important by either party by analyzing current SE curricula and learning outcomes from non-technical courses (Chapter 5) and conduct a skill gap study to compare all collected data so far (Chapter 6).

Part II

The Academia-Industry Skills Gap

In this part, we try to answer the **second RQ** of this work:

PhD Research Question 2

How big is the gap between the non-technical requirements of RQ1 and software engineering education?

Again, in order to answer this question, we conducted several separate works, which have been peer reviewed and published. First, Chapter 5 presents a study that scans learning outcomes of European universities for the presence of non-technical skills to support and strengthen the findings of the SLR from Chapter 3 (Groeneveld et al., 2020a).

Second, Chapter 6 presents a gap analysis that brings all gathered data so far together (Groeneveld, Vennekens, and Aerts, 2021b). Before that can be done, however, a translation had to be made to map the different encountered terminologies so far, as the results presented in the chapters of Part I produce different terms. This is also discussed in the second chapter of this part.

These chapters aim to shed more light on the gap between on the one hand what academia thinks are important non-technical skills—beyond a conventional literature analysis—and on the other hand what professionals think is needed to excel. The gap analysis reconciles data from all different sources and offers a clear view on the gap between both parties, answering `RQ2` in the process.

5 Teaching Trends: Program Syllabi

Publication details

This chapter has been published in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* as *Soft Skills: What do Computing Program Syllabi Reveal About Non-Technical Expectations of Undergraduate Students?* (Groeneveld et al., 2020a).
Its main contribution is providing an interactive database consisting of 278 non-technical computing-oriented syllabi from 110 European universities to have a better understanding of which skills are currently being taught in higher education, and which skills are still severely underrepresented.

In Chapter 3, we looked at what is currently being taught in SE education, based on published success stories. Papers that could not be found were obviously not included in the SLR. This only gives us a partial view of what is actually being taught.

In order to sketch a complete picture and to conduct a thorough academia-industry gap analysis, we first need to have an idea of which computing courses explicitly mention non-technical skills as part of their Learning Outcomess (LOs), next to the discussion of non-technical skills in job ads, interviews, accreditation and curricular guidelines, and other literature—these can all be

considered secondary sources, while we are interested in what is actually happening at universities.

In this chapter, we present a syllabi analysis of 278 manually curated syllabi from 110 universities in 30 European countries, where LOs were scraped from course information websites. 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.

Having established that non-technical skills are important and that model curricula encourage their inclusion in computing programmes, the question remains: How are these skills best incorporated into specific computing curricula? In other words:

Chapter Research Questions

1. *Which non-technical skills are currently taught in undergraduate computing courses, according to the learning outcomes?*
2. *What types of courses facilitate these skills the most?*

There is no related research that attempts to answer this question by scraping course information websites. Becker and Fitzpatrick (2019b) did analyze LOs of CS1 courses in search of expectations of introductory programming students but focused exclusively on technical hurdles (e.g. debugging, control blocks, inheritance, ...).

We first explain our approach to gathering syllabi data before presenting a summary of the results.

5.1 Collecting syllabi data

Our methodology was adapted from Becker and Fitzpatrick (2019a), who mined the 2016 QS World University Rankings® for Introductory Programming syllabi. A similar procedure was adopted, tailored for the aims of this study. We manually populated a database of 278 non-technical courses in 110 computing programs spread over 30 European countries. All

institutions were derived from the 2019 QS World University Rankings®, sorted by ranking, filtered on Europe. Countries not present in this list were not processed.

We deliberately narrowed down our search to only include European institutions for several reasons. First, we wanted to avoid introducing unintended bias—for instance by leading American universities which have a heavy presence in rankings.

Second, the Bologna accord (Fuller, Pears, Amillo, Avram, and Mannila, 2006), which most European countries are signatory to, ensures minimal equality between universities and the guarantees that courses (modules in Bologna terms) are based on explicitly stated learning outcomes, along with course weights expressed in European Credit Transfer and Accumulation System (ECTS). As evidence that this can affect searches such as ours, Becker and Fitzpatrick (2019a) found that their global search of syllabi, looking for learning outcomes, were skewed towards European Universities. Specifically they found more explicitly stated learning outcomes in syllabi from European countries. As the skills we are investigating are intricately related to learning outcomes, it is possible that these skills are reported more explicitly and more often than in other jurisdictions, due to the Bologna accord.

Third, we wanted a relevant and appropriately scaled data set. Since the authors are affiliated with European universities, they are more familiar with the European system. Since one of the authors can read four European languages, navigating websites became much easier. We believe that this methodical approach is representative enough to reliably answer our research questions, and also to provide a basis for comparison to other regions in the future.

During a feasibility study, we initially catalogued universities from UK, of which promising results encouraged us to continue populating the database. Since 30% of UK universities were covered, this coverage rate (rounded down) became the target for all other listed countries. However, some universities could not be included due to problems with data retrieval as detailed in the next section. We believe that a similar methodology could be applied to a global survey but there would be several issues that may

complicate the interpretation of findings including language barriers, the effects of different educational systems, and bias towards certain areas (such as the USA). Just as we found New Zealand (Stevens and Norman, 2016) and American-based (Aasheim, Williams, and Butler, 2009, Aasheim, Shropshire, Li, and Kadlec (2012)) studies to be helpful, we believe that a European-wide study will be beneficial to others.

Non-technical Course Database Population

CS syllabi were searched using the following steps:

1. On the institution website, locate the undergraduate computing information page, possibly located on the faculty site, or on the general program site. If the English website is incomplete, interpret the native language or use Google Translate.
2. Scan module lists and identify non-technical courses. Navigate to each such course syllabus and extract data.
3. If no syllabus link was found, perform a Google search for `(course identifier) + (university name)`. If this second attempt fails, exclude the university.

Identification of non-technical courses was done by excluding everything that is related to the theoretical, technical, and cognitive aspects of CS. For example, Aalto University in Finland offers a bachelor in Data Science[1], of which the curriculum contains typical courses such as "Matrix algebra", "Programming 1", "Databases", and "AI" that are not of interest for our study, while courses entitled "Creative Coding", "Introduction Course for Bachelor Students", "Introduction to Digital Business and Venturing", and "Professional Development" were selected for inclusion.

Since we wanted to focus on CS, only pure full-time undergraduate (bachelor) computing curricula were considered, excluding majors in engineering or science with minors in IT. Double degree programs such as Oxford's CS & Philosophy were also not taken into consideration. The following modules were also ignored: bachelor thesis, part-time courses, optional courses with course lists from other faculties, industrial placement or internships.

[1] See https://into.aalto.fi/display/enbsctech/Data+Science+2018-2020.

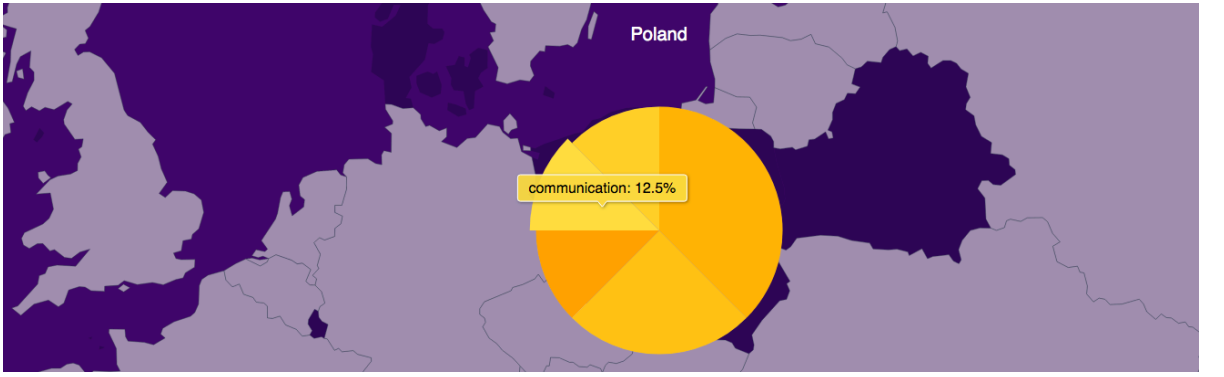


Figure 5.1: An interactive colour-coded map showing non-technical course categories in Polish universities. Our interactive data exploration tool is publicly available at `https://people.cs.kuleuven.be/~wouter.groeneveld/courses/`.

The following data was extracted from the link, when possible: university, program and course names, URL, year, optional, credits, learning outcomes. and ECTS Credits. Learning outcomes were saved with an English translation using Google Translate if required. Included courses were divided into the following colour-coded categories, as shown in the pie chart from Figure 5.1: art/creativity (1), coaching (2), business (3), professional skills (4), humanities (5), project-related (6), communication (7), and seminars (8).

Identifying Skills in Learning Outcomes

Course records were manually tagged with skills found in the learning outcomes. For example, if the learning outcome stated: "During the semester, students are required to work in teams to promote communication skills. At the end of this course, students should be able to: present their findings to an audience, and reflect upon their work." the following skills were tagged: *teamwork*, *communication*, *presentation skills*, *self-reflection*, while maintaining the original wording as much as possible. It is important to note that due to the ambiguity of these skills, some generalisations had to be made. Ambiguous terms such as *responsibility* and *accountability* have been grouped together. Courses without learning outcomes were not tagged. If learning outcomes explicitly mention skills beside cognitive knowledge, then the course will be

tagged as explicit.

To ensure the correctness of the identified skills and their categories, a cross-validation step was carried out, comparing and correcting data where needed. Due to the low number of differences and differing language abilities of the authors, no inter-rater reliability was calculated. We feel that this is justified due to the hand-curated nature of this study, and due to the large number of categories and other "hard" course facts (like the amount of credits) that were collected.

5.2 *Results & discussion*

We manually populated a database of 278 non-technical courses in 110 programs from over 30 European countries, covering at least 30% of the universities for each European country in the 2019 QS World University Rankings®. Table 5.1 summarizes the institutions included in our study.

Summary	#
Countries included	Austria, Belgium, Bosnia and Herzegovina Croatia, Czech Republic, Denmark, Estonia, Finland France, Germany, Greece, Hungary, Ireland, Italy Latvia, Lithuania, Netherlands, Norway, Poland Portugal, Romania, Russia, Serbia, Slovakia, Slovenia Spain, Sweden, Switzerland, UK, Ukraine
QS World University Rankings® `[highest]`	5
QS World University Rankings® `[lowest]`	range 801-1000
# Courses per institution `[mean (SD)]`	2.52 (1.43)
% Universities in each country `[max]`	100% (Bosnia and Herzegovina, Croatia, Serbia).
% Universities in each country `[min]`	15% (Russia)

Table 5.1: A summary of included results in this study.

If a given country only had one university we always included that. However, we excluded the following numbers of institutions for the listed reasons and included the next institution from the same country.

- No (pure) computing curriculum found (7)
- No computing undergraduate degree offered (6)

- No detailed course catalogue found (29)
- Website unreachable (1)

Thus 41% of the first encountered universities were excluded (and replaced). These are available on our website. This is better than the exclusion rate of Becker and Fitzpatrick, who excluded 77%. It is likely that their exclusion rate was higher than ours as they were searching for a specific course—albeit introductory programming being an extremely common one. It is also probable that they had more language difficulties than we had.

We created an interactive website for the community, available at `https://people.cs.kuleuven.be/~wouter.groeneveld/courses/`, to explore and visualise the data using a geographical map of Europe (see Figure 5.1), containing all data and results presented here (and more). The data can be searched and links to all syllabi are provided.

A compiled selection of noteworthy courses for each category from this section is also available online, together with help and information pages, as well as the ability to download the raw data.

RQ1: Which Skills are Taught?

Our first research question was: *Which skills are taught in undergraduate computing courses, according to their learning outcomes?*

Table 5.2 shows all identified non-technical skills[2], occurring in more than 5% of courses (278 courses total). This includes 17 of 50 skills total. The full list of skills can be found on the website.

[2] In Groeneveld et al. (2020a), we employed the term *soft skills*, but switched to *non-technical skills* since. See Chapter 2 for more information.

Communication (47.1%) and *teamwork* (34.2%) are the top two skills, largely agreeing with the literature and curricular recommendations reviewed in Chapter 2. Other skills agreed with prior work in being less frequent. For instance, *leadership* was mentioned in 6.1% of the courses we surveyed, ranked 29/48 by Aasheim et al. (2012) and featured in ~4% of job ads for Stevens and Norman (2016).

Surprisingly however, several skills that ranked highly in the SLR of Chapter 3 are featured very infrequently in our search. For example, *interpersonal skills* were mentioned in only 2.5% of the courses we surveyed, but was ranked 10/48 by Aasheim et al.

(2012) and in ~26% of job ads surveyed by Stevens and Norman (2016). Another striking difference is "Creative thinking", ranked 11/48 in Aasheim et al.'s (2012) summary, while *creativity* was only found in 4.7% of the courses.

Table 5.2: Non-technical skills identified in this search and the percentage of courses (n=278) including each skill.

Skill	%
Communication (written, oral)	47.1 (25.9, 21.2)
Teamwork	34.2
Ethics	26.6
Presentation skills	25.5
Role awareness	17.6
Critical thinking	16.2
Set/keep timelines	15.5
Self-reflection	12.6
Adjust communication to audience, Give/receive feedback, Problem solving, Conflict resolution, Leadership, Self-directed learning, Accountability, Effectively searching for data	5> each <10

It should be kept in mind that comparing these works is not straightforward. Aasheim et al. were ranking what they call "non-technical skills" identified by U.S. IT managers almost a decade ago. Stevens and Norman, although more recent, were looking at data from New Zealand only, and their job advertisement data only includes those ads that listed two or more what they call "soft skills". Nonetheless, broad trends such as the importance of communication and teamwork are apparently robust to different methodologies.

RQ2: What Types of Courses?

Our second research question was: *What types of non-technical courses are included in undergraduate computing programs?* We also asked: *What percentage of these courses are compulsory, in what year are they offered, and what percentage of them have explicit learning outcomes mentioning non-technical skills?*

All courses are grouped into eight categories, as visible in Table 5.3, along with the percentage of programs offering that type of course (%Prog.), the percentage that are compulsory %Compul.), the mean amount of ECTS credits received, the mean

year they are offered in, and the percentage that have explicit learning outcomes mentioning soft skills (%ELO). On average, courses take place in year 2, while students can earn 5.6 credits, of which 20.8% is optional.

Project based courses being "SE Team Project" and "Development project" harbor popular skills such as *teamwork*, *set/keep timelines* and *presentation skills*. This is the biggest group and the heaviest in terms of credits, and almost always compulsory.

Humanities based skills are *ethics*, *critical thinking*, and *role awareness*. This broad category represents courses such as "Philosophy", "Ethics in computing", and "Social aspects of IT".

Business based top skills include *teamwork*, *presentation skills*, and *leadership*. Examples of typical business courses are "Project management" and "Entrepreneurship". About half of the courses are optional, and these are usually taught in the later years.

Category	%Prog.	%Compul.	$\overline{\text{Credits}}$	$\overline{\text{Year}}$	%ELO
Project	31	93	7.8	2.1	37
Humanities	21	88	4.8	1.9	28
Business	13	54	5.0	2.3	35
Communication	10	66	4.0	2.0	28
Coaching	9	81	4.1	1.6	50
Professional	9	96	5.1	2.1	44
Seminar	3	66	3.0	2.3	11
Art-creativity	1	75	5.8	1.7	0

Table 5.3: % of programmes offering courses in each category, % that are compulsory, average credits & year offered, and % with explicit learning outcomes (ELO).

Communication based skills are *oral* and *written* communication, *presentation skills*, and *giving/receiving feedback*. "Public Speaking" and "Writing skills" are example courses of this category.

Coaching related top skills are *role awareness*, *ethics*, and *coaching* itself. Smaller introductory courses such as "Techniques of cooperative work" and "Studying at the University" are taught early in the program as these help students successfully graduate.

Professional skills include *teamwork*, *oral* and *written* communication, and *self-reflection*. Courses entitled "Professional Computing Practice" and "Transferable skills" are very interesting

to look at from a non-technical skill perspective, and are mostly compulsory.

Seminar based courses being "Selected chapters of Computer Science" and "Enterprise Seminar" teach *presenting*, *adjusting your communication to the audience*, and *written communication*. As visible in the mean credits, these are typically lightweight modules.

Lastly, **Art-creativity** houses skills such as *creativity* itself, *critical thinking*, and *intercultural skills*. The courses entitled "Bridging Arts and Technology" and "Creative Coding" sound very alluring to us, even if barely 1% of the programs offer such a course.

Only 33.7% of the courses explicitly mention skills in the learning outcomes, besides the requirement of pure knowledge-based attributes. Considering that we only included non-technical courses, as explained in Section 5.1, this is an alarmingly low rate. Many learning outcomes are in bad shape and contain nothing more than a few general sentences that make it hard for us as researches to analyse and for students to judge what is expected of them.

5.3 Threats to validity

Our study is not without threats to its validity. First, we intentionally only included European universities for three primary reasons, which we detailed in Section 5.1:

1. We wanted to get a representative picture, free from biases such as dominating US institutions.
2. Europe is largely unified by the Bologna accord (Fuller et al., 2006) and Becker and Fitzpatrick (2019a) (from which our methodology is derived) found that European institutions report learning outcomes explicitly (as the Bologna accord mandates).
3. Scale, relevance and expertise. The authors work and teach in different European countries and thus, this is where they are most interested, and familiar, resulting in a manageable, yet representative data set.

Another threat is that our results are based on world rankings.

We chose this for several reasons, many of which were justified by Becker and Fitzpatrick whose method we adapted, including:

1. Using rankings for similar purposes has been used successfully before, e.g. Guo (2014) and Becker and Fitzpatrick (2019a).
2. Using rankings does give a repeatable and "not biased by the researcher" method of sampling institutions.

Becker and Fitzpatrick also noted language difficulties as a problem that affected their sampling. Although we were not immune to this, limiting our search to Europe, and the fact that one of the authors is able to read four European languages, is a mitigating factor.

Another possible threat is that the list of skills for each course is incomplete or that some of the concepts from the learning outcomes are misinterpreted. To minimise this risk, a data validation step was added, as detailed in Section 5.1. The reliance on Google Translate for certain languages might also influence the skill list. Further data retrieval difficulties, as explained in Section 5.2, caused us to exclude some institutions. However, as these were replaced, a consistent ratio of universities for each country was maintained.

Finally, we acknowledge that this study is effectively a snapshot in time: academic syllabi are bound to change over time where soft skills may be added or dropped as required learning outcomes. It would be interesting to redo the study multiple times over the course of a few years to get an idea of how these learning outcomes evolve over time. We leave this to others as potential future work.

5.4 Conclusion

THE MAIN CONTRIBUTION OF this work is the finding that some non-technical skills are being addressed by computing curricula in Europe to a considerable degree, while others are in need of more attention. We also provide a web resource detailing what soft skills are present in European curricula, and where. Importantly, we also present data on what skills are infrequently taught. Additionally,

we find that the skills being addressed match prior work reporting on industry requirements—something we consider to be a positive development. Until now, however, this may not have been apparent as there was no clear and large-scale picture of the skills actually being taught, on what level, and in which courses.

The two most common skills found in literature—*communication* and *teamwork*—are consistently present in our results. Together with *ethics* and *presentation skills*, they form a strong and seemingly well-implemented core of non-technical skills in European computing curricula. It is equally interesting to investigate the other end of the spectrum, where skills/traits such as *creativity* and *empathy* are severely underrepresented.

This chapter, combined with the SLR of Chapter 3, can now serve as a baseline for the academia side of the academia-industry gap, while Chapter 4 represents the industry. In the next chapter, we will compare these findings and extract a list of skills that on the one hand professionals deem important, but on the other hand are underrepresented according to the SLR and LO analysis.

6 *Identifying the Skill Gap*

Publication details

This chapter has been published in *ACM Transactions of Computing Education* as *Identifying Non-Technical Skill Gaps in Software Engineering Education: What Experts Expect But Students Don't Learn* (Groeneveld et al., 2021b).
Its main contribution is mitigating the academia-industry gap by aggregating data from previous chapters and identifying non-technical skills that require immediate attention by the computing education research community.

So far, we have established that as the importance of non-technical skills in the SE industry increases, the skill sets of graduates match less and less with industry expectations. A growing body of research exists that attempts to identify this skill gap. However, only few so far explicitly compare opinions of the industry with what is currently being taught in academia.

By aggregating data from the SLR, the Delphi study, and the LO course analysis, we identify the three biggest non-technical skill gaps between industry and academia for the field of SE: *devoting oneself to continuous learning*, *being creative by approaching a problem from different angles*, and *thinking in a solution-oriented way by favoring outcome over ego*.

Eight follow-up interviews were conducted to further explore how the industry perceives these skill gaps.

Thus, the RQs that guide this chapter are:

Chapter Research Questions

1. *What are the most important non-technical skill gaps between academia an software engineering?*
2. *How are these skill gaps perceived by practitioners in the industry?*

We first provide background information related to work in the area before clarifying the utilized methodology and presenting the results of the gap analysis.

6.1 Related work

The quest to make an academic curriculum relevant to business requirements is certainly not a new one. One of the more iconic older publications, by Mize (1976) in the ACM SIGCSE Bulletin Journal of 1976, noted that "*in the past, the academic credentials have tended to indicate primarily theoretical training*". Several more recent works confirm that this is still the case today in the field of SE (Brechner, 2003; Jacobson and Seidewitz, 2014). Mize also found that:

> *Most studies that have been done in this area mention the lack of meaningful communication between the industrial community and the faculty members responsible for curriculum development* (*Mize, 1976*).

Since then, multiple efforts have been made to identify the skill gap between industry expectations and the abilities of graduates. Two recent systematic literature reviews stand out as being especially relevant: Radermacher and Walia's (2013) work, published in the 2013 ACM SIGCSE Proceedings, and that of Garousi, Giray, Tuzun, Catal, and Felderer (2019b), published in Volume 37 of IEEE Software in 2019. Both publications conclude that graduate students are indeed lacking in many areas, including technical abilities, personal skills and professional qualities. However, they do not focus exclusively on non-technical skills. Also, knowledge and competence deficiencies are identified by counting occurrences of keywords in papers, not by explicitly comparing opinions of the industry with opinions of the academic world, as we will do.

Radermacher, Walia, and Knudson (2014) continued to investigate the skill gap in 2014 by interviewing 23 practitioners in the field across the United States and Europe. The results of their study are again quite broad: graduates seem to be struggling with using various technical software tools, effectively communicating with co-workers and customers, producing unit tests for their code, and other skills or abilities that are not further specified.

Moreno, Sanchez-Segura, Medina-Dominguez, and Carvajal (2012) investigated the SE curriculum knowledge guidelines provided in *Software Engineering 2004* (SE 2004) (Lethbridge, LeBlanc Jr, Sobel, Hilburn, and Diaz-Herrera, 2006) and *Graduate Software Engineering 2009* (GSwE 2009) (Ardis, Bohner, Fairley, Towhidnejad, and Pyster, 2009), and the job profiles identified by Career Space. They conclude that none of the three job profiles ("*Software and Applications Development*", "*Software Architecture and Design*", and "*IT Business Consultancy*") is completely covered by either SE 2004 or GSwE 2009. Of course, the noted lack of certain soft skills has since then been partially mitigated in newer iterations of these curriculum guidelines (Ardis, Budgen, Hislop, Offutt, Sebern, and Visser, 2015).

Liebenberg, Huisman, and Mentz (2015) investigated the industry's perception of the relevance of software development education, using a mixed methods approach. The SE experts made proposals such as "*lecturers should have industry experience*" and "*people from industry should be brought into software development classes*". Liedenberg's team conclude by suggesting to devote more attention to, among others, teamwork, projects and experiences, soft skills, career guidance, and introspection.

To the best of our knowledge, an in-depth analysis of non-technical skills in detailed terms (e.g. "*understanding and engaging with the people involved*") rather than high-level categories ("*communication*") is still missing. The contribution of this chapter is to provide such an analysis. Furthermore, the current literature presents only one side of the story: that of the industry expert, dissatisfied with abilities of graduates. We aim to approach `RQ1` from two sides: the side of industry (interviews with practitioners, data from the Delphi study), and the side of academia (current

state of affairs in teaching non-technical skills, data from the SLR and course analysis).

Interestingly, there is also literature on why we as educators should opt to teach what industry distinctively does not want, as opposed to inspecting RQ1 from two sides. For instance, Ryan (2020) mentions that taking ethics into consideration is rarely something a commercial company is after. Indeed, we also found ethics in both the SLR and course analysis studies, but it was not mentioned at all in the Delphi study while interviewing industry experts. However, the focus of this study is to approach the problem from both worlds, not from one.

6.2 Methodology

This research makes use of the data presented in the previous chapters, more specifically the 2019 SLR study from Chapter 3, the 2020 LO study from Chapter 5, and the 2020 Delphi study from Chapter 4 (identified as DEL). For more information on these studies, please see their respective chapters. The context and relation to the study presented in the current chapter is visualized in Figure 6.1.

In order to combine data from all three previous studies, we need to map terms across different studies. For example, in our SLR, skills such as "*communication*" and "*self-reflection*" were identified, while in the Delphi study, experts identified "*actively listening*" and "*understanding and engaging with people involved*". The Delphi results will serve as the reference skill set as these were verified by software developer veterans from the industry. Thus, we will link skills currently being taught (SLR and LO) to those that are required (DEL).

Research question RQ1 can be answered after mapping and aggregating skill data. To further explore how this identified skill gap is perceived by the SE industry (research question RQ2), semi-structured follow-up interviews were conducted. These interviews complement and reinforce our findings, but do not serve as a primary data source for answering RQ1.

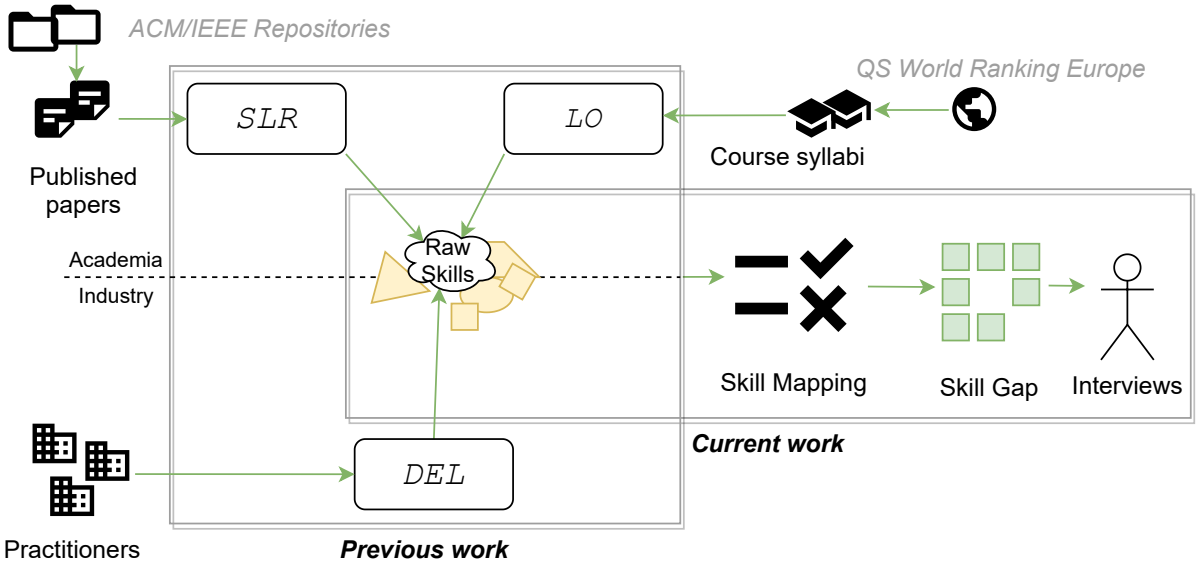


Figure 6.1: The context of previous and current chapters.

The Skill mapping exercise

Different papers use different sets of SE topics: some use versions of SWEBOK (Bourque et al., 2014), while others use the "SE2014" curriculum guidelines outlined by ACM and IEEE (Ardis et al., 2015). Mapping SE terms has been done before by Garousi et al. (2019b) in their SLR. However, they fail to mention how exactly skills were mapped. In another study by Tariq, Scott, Cochrane, Lee, and Ryles (2004), experts have been interviewed to audit and map generic skills development across a UK degree programme. We based our methodology on this paper.

We developed an on-line mapping tool to make the connection between skills identified in our different previous studies. We solicited the help of eight researchers to use this tool to make the necessary links. These volunteers have ample experience with educational research on skills without being biased as a participant of the previous studies.

Figure 6.2 shows the tool in action. Researchers can drag skills from left to right to create a link between both sets of non-technical skills. Both one-to-one and one-to-many relations are allowed. In

Figure 6.2, a connection from '*communication*' to both '*adjust your communication to the target audience*' (red link) and '*actively listen*' (green link) was made. Of course, '*communication*' is a high-level skill that can contain multiple communication-related sub-skills. After skills have been mapped, a "Save links" button persists the data in JSON format for further processing.

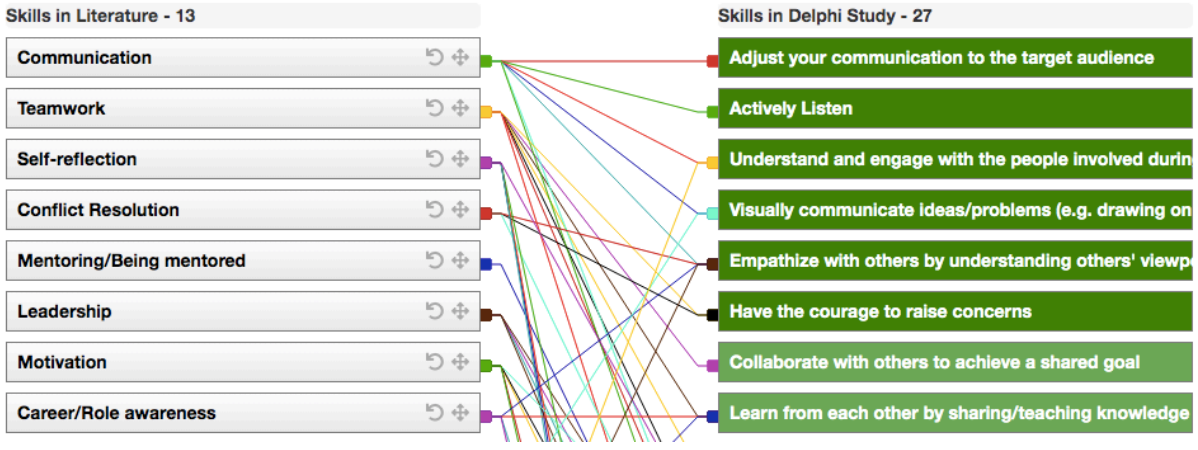


Figure 6.2: An illustration of a skill mapping exercise in progress: from SLR skills (left) to Delphi skills (right). Researchers can drag skills to create a link. For example, has been linked to six different Delphi skills.

Aggregating skill data

After collecting data from the mapping exercise, a few normalization steps were needed to be able to compare occurrence data from all three studies.

In order to compare data of our different studies, we first convert the raw counts of how often each skill occurred into a percentage. We first explain how this was done for the Delphi study[1]. Here, we opted to use the data from the skill selection round (round 3). This data pruning round reduced the skill set from 55 to 27 relevant skills, as Delphi participants pick 10 factors they consider to be the most important skills for a developer to excel in the SE industry.

For each panel P ($P \in \{T, B\}$), we denote the size of P by N_P. In round 3, each of the participants indicated which skills they found important. For panel P, we denote by $n_P(s)$ the number of participants in P who flagged skill S as important. We then define

[1] Only the data from the technical and business experts panels (panels T, B) has been utilized, as these industry veterans are most relevant to our research questions, and the other two studies already contain data from academia. See Chapter 4 for more information.

the percentage $X_p(s)$ as $\frac{n_P(s)}{N_P}$.

We choose to base ourselves on round 3 of the Delphi study because data from this round most closely resembles the data from our other two studies, which also counts how many times skills are mentioned in a certain context. However, round 4 of the Delphi study also contains useful information, which we like to incorporate as well. For this reason, we will weigh the occurrence percentages $x_P(s)$ by a weight that is derived from round 4. In this round, panelists were asked to rank all of the skills that were carried over from round 3. For panel P, we denote by M_P the number of skills that were ranked by panel P and by $r_P(s)$ the rank of skill s. We then define the weight $w_P(s)$ of skill s in panel P as $\frac{100}{M_P-1} \times (M_P - r_P(s))$ for those s such that $r_P(s) < M_P$. For the skill s that was ranked in last place (i.e., $r_P(s) = M_P$), we define the weight $w_P(s)$ as 1 to avoid entirely collapsing its final score. The score p_s of a skill s is then defined as:

$$\frac{\sum_{P\in\{T,B\}} w_P(s)x_P(s)}{\sum_{P\in\{T,B\}} w_P(s)} \tag{6.1}$$

For example, the Delphi skill "*Being creative*" occurred 7 times in round 3 for panel T, where 9 panel members voted. Expressed as a percentage, $x_P(s) = (7/9) \times 100 = 77.78$. To determine the weight, we use the data from round 4, in which creativity ended third out of 18 selected skills by panel T. Thus, $w_P(s) = \frac{100}{18-1} \times (18-3) = 88.24$. We repeat the same for panel B, where creativity occurred 4 times with 10 panel members, and where it ended at place eight out of 14 selected skills. After summarizing and averaging, we can conclude that $p_s = 64.80$, as visible in Figure 6.3.

The skills from SLR and LO are mapped onto DEL skills. For each of these skills, we consider the percentage of papers (SLR) as courses (LO) that refers to the skill. For instance, '*Self-reflection*' was identified in 13 out of 26 papers in the SLR study, which is 50% of the time. For each connection made during the mapping exercise, the average of all percentages of SLR and LO skills that were mapped to the same DEL skills was calculated. We will compare this average percentage to the DEL score as defined above.

Follow-up interviews

Besides the identification of the non-technical skill gaps (RQ1), we were also interested in how the industry perceives these gaps (RQ2). Semi-structured interviews were conducted to provide more information about this topic. The interview selection process was adapted from a study by Li et al. (2015), where industry veterans were approached based on their achievement of some degree of recognition as SE experts.

We limited participation to people with a technical role who come into daily contact with source code, such as *developers*, *programmers*, *software architects*, ... Moreover, experts were required to have at least seven years of experience in the field, and should be intrinsically motivated to partake in the discussion. After carefully reviewing their expertise based on information at hand, such as blogs, CVs, and social media, experts were invited until thematic data saturation was reached. We followed the guidelines of Guest, Namey, and Chen (2020) and Marshall, Cardon, Poddar, and Fontenot (2013) to achieve this, as discussed in Section 6.3. It is important to note that previous respondents from the Delphi study were excluded to avoid any bias or redundant results. All participants were Caucasian male Belgians and active software experts in the field of enterprise SE, as this is the same context as the Delphi study.

The conversations were facilitated by the author of this thesis, since he has more than 11 years of experience in the industry. Even if some researchers claim that familiarity with the topic will introduce bias (Fern and Fern, 2001), Kontio, Lehtola, and Bragge (2004) state:

> *In the field of SE, it is very important for the interviewer to have extensive knowledge of the theme of the interview.*

All interviews were conducted in Dutch to keep the conversation fluid, as this was the native language of all participants. We do not believe that the language limitation negatively affects our results, as the Delphi survey was taken across 11 different countries world-wide, and our primary aim for the interviews was to shed new light on these results. Companies

involved differed in size, from national to internationally renowned. Again, only enterprise software development companies were taken into consideration, as was the case with the Delphi study.

The interviewees were approached via e-mail, where information about the design of this research was provided, including the graphical results of the gap analysis (see Section 6.3). A digital meeting took place after each expert gave his consent to anonymously record the conversation. At the start of the interview, the design and purpose of the study was briefly summarized again, including the top three largest gaps that were identified. We continued with the following questions:

1. *Do you perceive a possible lack of continuous improvement in your company?*
2. *Do you perceive the possible lack of creativity as a problem?*
3. *Do you perceive solution-oriented thinking as a possible problem?*
4. *How does your company deal with the possible lack of these skills at the moment?*
5. *How do you think higher education should address this skills gap?*

Furthermore, we asked follow-up and clarification questions about responses we found intriguing (e.g., novel, vague, avoiding or counter to prior answers). The questions were formed in accordance with Hove and Anda's (2005) guidelines for conducting semi-structured interviews in empirical SE research, which state that one should "*Ask experience questions, avoid 'why'-questions and questions with 'yes'/'no' as answers*". The interview was closed by restating the purpose of the research, including future work, and by asking if the interviewee had anything to add.

The collected data from the interviews was processed using qualitative coding as presented by Onwuegbuzie, Dickinson, Leech, and Zoran (2009) and McAlister, Lee, Ehlert, Kajfez, Faber, and Kennedy (2017). First, the audio recordings were transcribed by the author of this thesis. Next, a first iteration of a codebook was developed through open coding techniques, by re-reading the transcripts multiple times. This initially resulted in a set of 53 codes, which was reduced to 33 codes by grouping similar terms.

The goal was to develop a codebook that could be easily used by all authors to consistently code the transcript documents. Each code in the codebook is accompanied by a clear definition and an excerpt of an interview where the code is applicable, grouped into six bigger themes. Then, each researcher coded the first interview independently, after which the results were thoroughly discussed. This resulted in an initial average inter-rater reliability (IRR) of 26% for each code. For this, the IRR formula mentioned in McAlister et al. (2017) was used. The codebook was changed to clear up any misconceptions, resulting in the final set of 26 codes. Next, using this revised codebook, all researchers coded the second interview. This resulted in an initial IRR of 50%, which is a clear improvement over the first IRR. Further discussions between the authors revealed that the disagreements were due to varying interpretations of code definitions. Finally, after reconciliation of all differences and updating the codebook again, all authors fully agreed on the codes of the first two interviews, resulting in an IRR of 100%. The rest of the interviews were coded by the author of this thesis, using the final revision of the codebook.

6.3 *Gap analysis results*

From the Delphi study, the 27 skills that made it through round three were used as the target to map the 13 different skills found in the SLR and the 50 skills found in the LOs. Eight experts helped us with this mapping exercise. After applying normalization steps explained in Section 6.2, the results were arranged in the skill comparison chart shown in Figure 6.3.

We are interested in the biggest gap between what the industry thinks is important (*purple bar*) and what is currently being taught (*yellow bars*). A positive delta denotes a higher demand from the industry compared to what is currently embedded into the curriculum, while a negative delta denotes that we as educators could possibly be putting too much emphasis on that particular skill. The same information is also visualized in Figure 6.4, which depicts the calculated delta values for each skill, categorized in the

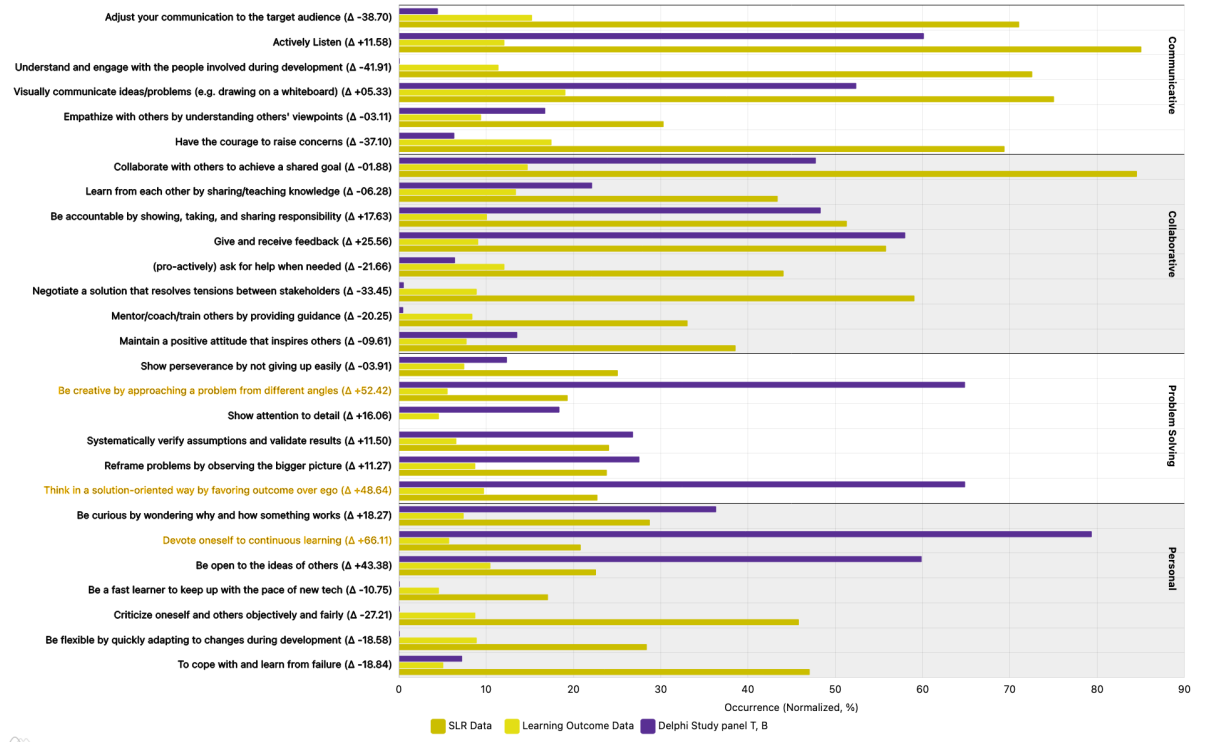


Figure 6.3: The skill comparison chart, comparing data from the Delphi study (bar 1, in purple) to Learning Outcomes (bar 2) and SLR findings (bar 3). The bigger the difference between the yellow and purple bars, the bigger the perceived skill gap, as also indicated by the average delta values. An interactive version is available at `https://people.cs.kuleuven.be/wouter.groeneveld/gap/`.

same way as in the Delphi study. The groups '*Problem solving*' (5 out of 6, 83%) and '*personal*' (3 out of 7, 43%) have particularly high delta values, compared to the groups '*communicative*' (2 out of 6, 33%) and '*collaborative*' (2 out of 8, 25%).

The three largest "positive" skill gaps, as emphasized in dark red in Figure 6.4, are *devoting oneself to continuous learning* (δ +66.11), *being creative by approaching a problem from different angles* (δ +52.42), and *thinking in a solution-oriented way by favoring outcome over ego* (δ +48.64). These skills are further highlighted in Table 6.1. The three largest "negative" skill gaps are *understanding and engaging with the people involved during development* (δ -41.91), *adjusting your communication to the target audience* (δ -38.70), and *having the courage to raise concerns* (δ -37.10).

We conducted eight semi-structured interviews, which lasted on

Skill	DEL occ.	δ SLR	δ LO	δ Avg.
Continuous learning	79.32	59.99 (19.3%)	73.92 (5.40)	66.11
Creativity	64.81	49.81 (15.0%)	54.01 (10.80)	52.42
Solution-oriented thinking	64.80	49.80 (15.0%)	59.40 (5.40)	48.64

Table 6.1: The top three skill gaps with deltas for each study, normalized and expressed in percentages. The average delta (δ Avg.) is the average of the SLR and LO percentages, compared to the DEL occurrences.

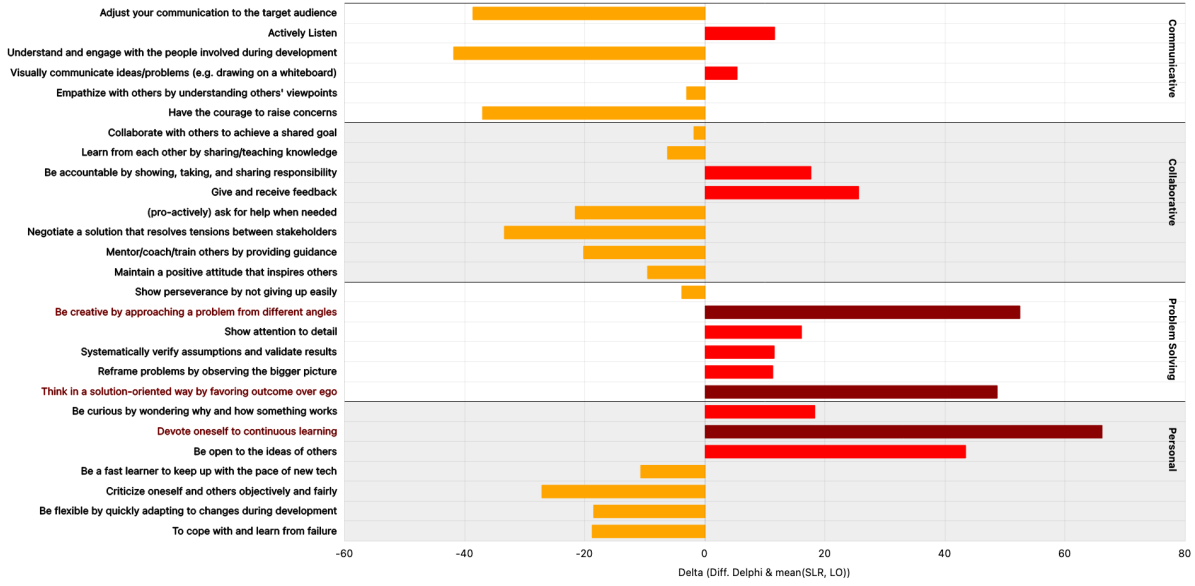


Figure 6.4: An alternative view depicting deltas for each skill group identified from the Delphi study. A positive delta ((dark) red) denotes a higher demand from industry compared to what is currently taught, while a negative delta (orange) denotes that the skill is included more in academia than in industry.

average 38 minutes. In total, 303 minutes were recorded, yielding 43604 transcribed words. Table 6.2 displays interview paradata, including the discovered theme distribution of the identified codes that was used to verify data saturation levels. To achieve this, we followed the guidelines of Guest et al. (2020). After the first four interviews, we counted the amount of identified codes (23). Next, two more interviews are conducted, and the codes for the new interviews are counted again (2). This means 9% new codes were introduced (changed over base), which is not small enough to stop the process. The process repeats itself until the pre-set threshhold of 4% has been reached, after the seventh interview. Lastly, one more interview was conducted to verify no major change in the threshold was introduced.

In the following section, we will briefly discuss each skill listed in

Description	Int. 1	Int. 2	Int. 3	Int. 4	Int. 5	Int. 6	Int. 7	Int. 8	Total
Interview length (min.)	32:33	37:14	30:50	35:24	42:49	35:06	44:51	44:07	05:02:54
Transcribed words	4869	5430	4661	4844	5841	5302	7159	5498	43604
New themes in int.	14	6	1	2	1	1	0	1	26
New themes in run				23		2	1	1	/
% change over base						9%	**4%**	**4%**	/
Discovered theme distr.	54%	77%	81%	88%	92%	96%	96%	100%	/

Table 6.2: Interview paradata, including saturation assessment data, using a base size of four interviews and run length of two. An information threshold of $\leqslant$ 4% indicates data saturation.

Table 6.1, both in context of existing literature and in context of the conducted interviews.

6.4 *Interview results*

IT IS CLEAR THAT all three skills from Table 6.1 are linked: there is no creativity without learning new ways to express yourself, and there is no solution-oriented thinking without a bit of creative work. During the interviews, the same kind of initiatives were frequently mentioned as a way of enhancing all three skills. In total, we identified six overarching themes: *stimulating continuous learning*, *stimulating creativity*, *creative techniques*, *addressing the gap in education*, *skill requirements in industry*, and the *industry selection process*. These six themes can be further divided into 26 subthemes, as summarized in the mind map in Figure 6.5.

In the discussion that follows, the subthemes we discovered have been marked in bold. Wherever possible, and for the purpose of maintaining readability, discussions of the skills where interviewees had substantial comments about have been divided into their own subsections. The other two themes, *skill requirements in industry* and the *interview selection process*, were discussed only briefly by the interviewees. They are summarized at the end of this section.

Continuous learning

Continuous learning is pushing the boundaries of your own knowledge and skills, every single day. According to most interviewees, it is the heart of SE. This aligns with the existing

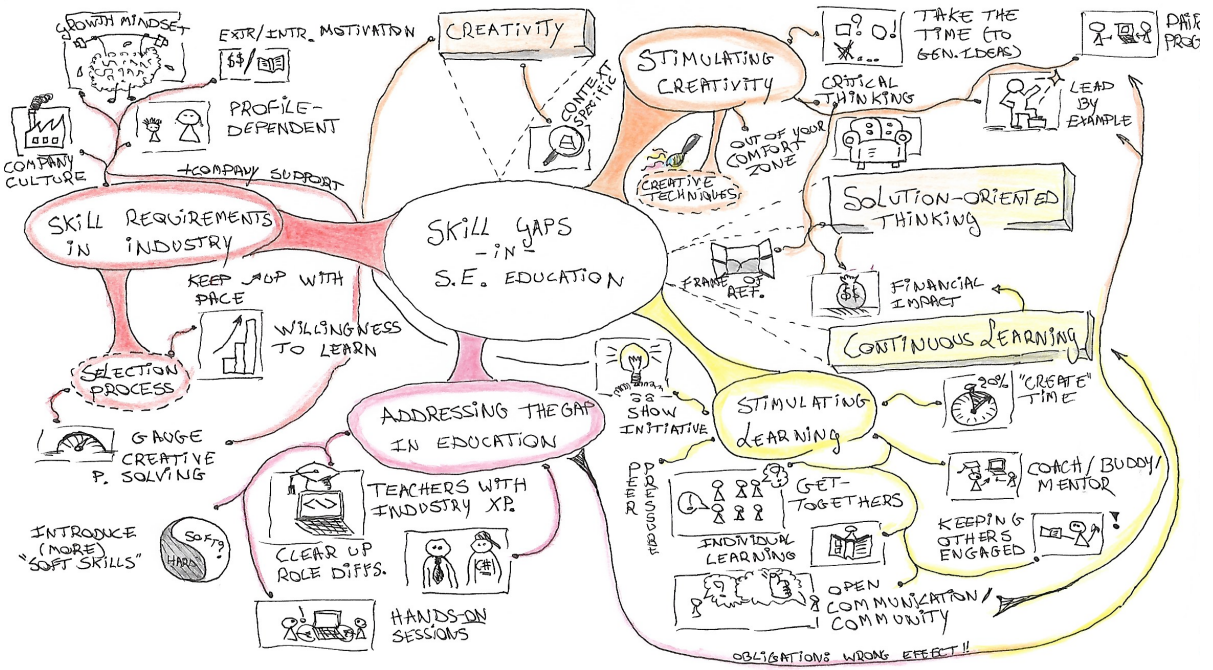


Figure 6.5: A mind map that summarizes the themes we identified.

literature. Participants in a survey conducted by Exter, Caskurlu, and Fernandez (2018) universally agreed that critical thinking, problem solving, and especially lifelong learning are the most important skills that are essential to their job as software developers. During our own interviews, a participant warns that without a **passion** for SE, you are not going to make it:

> *The most important thing is to motivate people to learn new things. Nourishing the curiosity flame, so to speak.*

When asked how to keep that flame burning and how organizations deal with a possible lack thereof, the panelists present several initiatives, that are **supported by the companies** they work in:

1. Attending conferences, **participating** in Software Crafting and Testing (SoCraTes) meetups and lean coffee events

2. Engaging in reading groups, architecture forums, or 'brown bag' sessions, where new technologies are being discussed
3. **Individual learning** using platforms such as Udemy or Pluralsight, or through classic internal or external training programs
4. Participating in 'sandboxes' and competence centers: small groups of like-minded people with a central technical theme that try out new technologies.
5. Following well-known software developers on social media or websites such as DZone[2].
6. Following internal Slack channels where interesting leads are posted and new technologies to pickup are discussed

[2] See https://dzone.com.

One participant came up with a colorful analogy, originating from the "soft skills" of Sonmez (2016): a developer should see himself as a shop window. You can display your freshly honed skills in it, but if not kept up-to-date, things will deteriorate and become dusty quickly. Nobody likes shopping in a store where the window hasn't been regularly cleaned.

It is striking that the above initiatives almost always take place in groups, with a strong focus on **open communication**. Some companies even employ a person who dedicates one day a week to learning and **keeping others engaged** in it. Senior **coaches** are frequently assigned to junior developers to help them get acquainted with the company's lifelong learning vision. The coach follows up the learning progress and constantly pushes newcomers **out of their comfort zone**. This is decoupled from the classic yearly evaluation interviews, as the casual chats feel less intimidating and happen more frequently, providing early feedback that helps juniors stay on track.

The use of **pair programming** is also often mentioned. This, combined with the coaching system, feels very similar to apprenticeships: one-to-one long-term intensive guidance that ultimately breeds the right attitude towards continuous learning.

That attitude should also be put to good use: without the proper **company culture**, it will fail to flourish. A participant made an analogy with growing plants:

The seed may yet be very powerful, but it cannot grow without fertile soil

and support from the company that's planting it.

Peer pressure is another consideration. Many interviewees reported that at one point in their career, they started learning new things "because others did too". If everyone learns, so do I. A stimulating vibe arises: during lunch, people talk about various initiatives, sparking the interest of others. This was reportedly especially relevant to motivate people who weren't already.

However, as mentioned before, there seems to be a need for a base level of intrinsic motivation. A pilot light should be present that, with coaching and the right company culture, can be ignited into a bigger flame. That is why, according to interviewees, very **strict selections** are made during applications, focusing almost entirely on the willingness of the applicant to learn new things. Half of the participants (4 out of 8) confirmed the lack of continuous improvement in newcomers while others (2) said that due to these demanding selections, their company did not have this problem.

Everyone agreed that as a software developer, it is essential to continuously improve one's knowledge. Yet, two participants noted that sometimes it is not very clear what exactly to learn. Technologies come and go at a rapid pace, and sometimes one invests in the wrong framework that loses traction in the development community or gets replaced by something else. Therefore, it is also important to gain the right knowledge. Reports from prominent industry leaders, such as the popular ThoughtWorks® Technology Radar[3], can help guide developers in the right direction.

[3] See https://www.thoughtworks.com/radar/.

Creativity

When looking at existing research into the skill gap by means of expert interviews, it is striking to see that creativity is not explicitly mentioned anywhere (Exter et al., 2018; Garousi, Giray, and Tuzun, 2019a; Radermacher et al., 2014). Bailey (2014) mentions that "software developers are considered to be the creative minds behind computer software", yet does not list creativity anywhere. We can only guess that it is implicitly included in problem solving,

which they report as the most important non-technical skill for software developers and business analysts. By contrast, in our interviews we dug deeper to explore the meaning and context of creativity.

Interviewees complained that many graduates lack the creative insight needed to solve complex problems. To mitigate this, coaching and pair programming again were mentioned, along with code reviews that stimulate open discussions, broadening each others' worlds instead of simply copy-pasting Stackoverflow code. In short, coaches try to teach newcomers various **creative techniques** to facilitate problem solving, such as zooming in and out of code to approach things in new ways.

People too often go back to things they know, without thinking what could be done differently. Developers are encouraged to use the whiteboard to draw and discuss pieces of code, and to think out of the box by asking the "why" question. As one participant said:

> *To me, creativity is daring to question things. Having a **critical mindset**.*

As an example to prove his point, he referred to the typical horizontal slices of most enterprise software architectures (front-end, service-layer, back-end), compared to functional vertical slicing. Instead of conforming to the widely used "anti-pattern standards", he introduces alternatives by letting teammates watch presentations of Jimmy Bogard's vertical slice approach[4].

[4] See https://jimmybogard.com/vertical-slice-architecture/.

The best way to show how things can be solved differently, is **leading by example**. By stimulating more discussions and by encouraging juniors to **take their time** instead of jumping the gun, a **frame of reference** is built up that can be leaned upon for future challenges. "In the beginning, we train people", one participant explained. "We teach 90%, and 10% is the unknown for them to discover." After gaining some experience, the trainer becomes the coach: "5% is shown, 95% left open to learn themselves." This is especially relevant for new frameworks and libraries, where the continuous learning attitude should kick in and take over.

However, one should not cling to that frame of reference for too long. Every problem is **context-dependent**, and developers should have the creativity to break free from general best practices within

a specific context, as one participant noted. What will work in one project, might not be the best solution in another.

Many interviewees cannot make a clear distinction between the discussed skills, confirming again that all three skills (continuous learning, creativity, solution-oriented thinking) are somehow tightly coupled. In literature, where 'creativity' is found, (some form of) 'continuous learning' also appears (de la Barra, Crawford, Soto, Misra, and Monfroy, 2013; Garousi et al., 2019b; Li et al., 2015). de la Barra et al. (2013) go as far as saying that "Agile Software Development is about knowledge management and creativity". According to interviewees, without continuous learning, there would be little creativity, as learning both technical domain-relevant expertise and creative skills is needed in order to creatively solve problems. This is consistent with Amabile's (1988) Componential Model of creativity.

Besides a need for both a learning and a critical mindset, there should also be time to support experimenting:

> *My conclusion is that you should give freedom. And then good things will come.*

That freedom can be seen in companies organizing hackathons with **open assignments**, unrelated to the commercial agenda of the company. These social events also provide great opportunities for like-minded people to pair up, in case they want to build something outside of company time. It seems that, according to the interviews, the most highly regarded companies are the ones who implement a weakened form of Google's famous (and abandoned) "20% time".

In another interview, a participant again emphasizes the social aspect of SE. "Programming has become a real team sport", he explained, ending the interview with the comment:

> *Too little communication is too little creativity.*

Creative problem-solving techniques

The skill labeled as creative problem-solving techniques (in the Delphi study denoted as "solution-oriented thinking") frequently appears at the top of skill gap lists, such as Radermacher et al.'s (2014) and Exter et al.'s (2018). However, in both of these works,

there is no mention of creativity, or of how to actually tackle software problems. In our Delphi study, we viewed problem solving as one of four categories, consisting of multiple skills, rather than as an individual skill. For instance, "being creative by approaching a problem from different angles" and to "think in a solution-oriented way by favoring outcome over ego" both are part of this category.

Nearly all remarks made during interviews surfaced both in context of problem solving and of creativity. Software engineers use creativity to solve problems, so naturally, "solution-oriented thinking" became a synonym for "creative problem solving". It appeared to be too difficult for participants to make the distinction between these two skills, even when we probed them with additional questions to try and identify the difference. Also, participants seemed much more eager to talk about creativity. Therefore, we decided against making an artificial distinction.

Addressing the skill gap in higher education

A few distinct issues can be identified when analyzing the answers to the question of how higher education can address the skill gap. To begin with, many interviewees complain about the fact that education lags too far behind on the technology that is being taught. This gives the impression that schools are not learning themselves:

> *If schools are not involved* [*in continuously learning new technologies*]*, how can you expect students to do that themselves?*

Teaching older Java technologies such as JSF and JSPs are some examples. This frustrated many participants, to the point that a few even claimed to prefer graduates not to have any knowledge at all, compared to the **wrong kind of knowledge**. it is apparently difficult to "unlearn" older concepts, in favor of newer ones. According to interviewees, learning the wrong kinds of frameworks is not only useless, but it makes learning the "right" frameworks more difficult.

To mitigate these problems, participants propose that **lecturers should have more industry experience**, as also proposed by Liebenberg et al. (2015). The shop window analogy also applies to

the skills and knowledge of teachers. The best way to induce an attitude of lifelong learning in students is by embodying it yourself. Another proposed initiative is teaching students how to filter what to learn and what to ignore. Especially in the world of software development, programming languages and frameworks evolve at such a rate that it's simply impossible to learn everything.

As mentioned before, existing gap analysis studies do recognize lifelong learning and problem-solving skills as important. Interestingly, Exter et al. (2018) mention that identifying the skill gap is not the whole story. They found that "*conversations with faculty and administrators indicate that they feel that they are covering many of the crucial areas identified by professionals, but that students often do not appear to recognize or retain what they have learned*". According to the authors, there seems to be a large discrepancy between graduates' and faculty members' perceptions.

Our interviewees seem to agree that although many prevalent non-technical skills are being taught in higher education, current initiatives to **promote lifelong learning are not very effective**. Participants remember that during their education, they had to partake in initiatives such as joining meetups or conferences. However, it felt too obligatory, such as reading required books in high school, and in the end most students simply do these things to tick them off the required list. The proper context is difficult to get right in a limited school environment. In the end, the notion of lifelong learning is thereby lost. As an alternative to these current practices, interviewees propose to use this time to let students struggle with learning a programming language they are not familiar with. This more accurately simulates their future work environment: rapidly learning new things within a limited time-frame. Of course, the question remains whether students would not also consider this as merely another compulsory item to tick off.

The following quote neatly summarized various discussions regarding the topic:

> *Who do students look up to? To someone knowledgeable, someone that inspires them.*

One can inspire students with live programming sessions, using

keyboard shortcuts to let students get a taste of the power of productivity tools, and by actually showing things working, instead of reading boring slides. This is the "demo or die" mantra from Scrum and the MIT Media Lab (Sutherland and Heitz, 2011) applied to study material in the classroom.

Another suggestion is to organize code kata sessions, using mass pair programming by passing on the keyboard ("swarming") (Arsenovski, 2016). Showing enthusiasm and experience will hopefully grab students' attention and inspire them, in keeping with participants' suggestions to lead by example.

Most solutions that were proposed to mitigate shortcomings are clearly geared towards **hands-on sessions**. Many participants complain that there is too much theory and too little emphasis on the hands-on "feel". As a result, introducing more project-based work has also been often suggested. In recent years this suggestion has been taken to heart in higher education. However, there is a lot of room for improvement: there is still barely any explicit mention of **soft skills** or any synonyms in Learning Outcomes, as also noticed by participants.

Summary

Interviews showed a similar pattern when the question of how companies deal with possible shortcomings in skills was discussed. First, developers should **show initiative** and indicate themselves that they both want to learn and can in fact **keep up with the pace** of constant innovation. Next, this learning can be reinforced by the support of colleagues and the company, but the initial attitude towards continuous improvement should be present.

An interviewee mentioned Dweck's (2008) "growth mindset" system: the application process acts as a "fixed mindset" filter. According to participants, it is very difficult, if not impossible, to introduce changes into a group of "fixed mindset"-developers. To do that, a small group of **change makers** should join forces. One must radiate some kind of passion, that perhaps does not need to be SE-oriented. That very same passion also helps tremendously when leading by example to ignite other's passion flame. Unfortunately, interviewees also report that people with great

intrinsic interests are in short supply.

As mentioned before, communication seems to play a big role in nourishing the three skills:

The human aspect in software development is very important.

Pair programming is a good example of this. However, it should be noted that one participant mentioned that sometimes pairing up seniors with graduates does not work well because the knowledge gap is simply too big. As a trainee, it is sometimes better to process new information by yourself, at your own pace.

To summarize, the discussions in this section contribute to the identification and analysis of non-technical skill gaps in computing education. Our data analysis shows this gap from two perspectives. On the one hand, it displays the current state of affairs in teaching these skills in academia using data from the SLR and learning outcomes. On the other hand, we investigated the perspective of industry with the help of interviews with practitioners and data from the Delphi study. This approach is unique, as we found no evidence in literature that presents data from both sides of the story. Furthermore, the discussed skills in the Delphi study and the discovered themes in the interviews are very concrete terms. This allows educators to more easily implement them into their curriculum. A few important questions remain, however: which of the proposed practices would work in academia? Which of the suggestions are new or unique? Future work could delve into these details to develop practical proposals for curriculum makers.

6.5 *Threats to validity*

SINCE WE AGGREGATED DATA from our previous studies to answer the research questions of this study, possible limitations for each study also apply here.

A direct comparison between results of the Delphi study and skills currently being taught might lead to incorrect conclusions. During the Delphi study, we asked experts to list skills needed to *excel in the SE industry*—we did not ask for minimum requirements.

This was done to receive more pronounced answers, and might slightly affect the mentioned skills or the ranking of the skills. Also, as some industry experts rightfully remarked, skill requirements can change depending on where you are in your career.

Another possible threat to the validity of our results is that the list of skills identified in the course analysis study might be incorrect or that some concepts from the learning outcomes are misinterpreted. Skills that are taught but not explicitly listed in the learning outcomes are of course not included, possibly resulting in a low bar in Figure 6.3. The same holds true for the SLR. Suppose there are no publications about continuous learning because students are already engaged in it, since they learn new material each day. If that were the case, it would not appear in the results of the SLR. To mitigate these issues, we combined data from both the SLR and LO, instead of relying on one data source from the academic world.

It could be that the two populations matched when looking for gaps between industry needs and what is being taught is not perfectly aligned. For instance, learning outcomes are extracted from European universities, while the Delphi study includes experts from outside Europe. This potential mismatch is minimal, however, since 86% of the industry expert panel is European[5].

[5] More information on the demographics of the Delphi participants is available in Chapter 4.

Also, semantic problems might have arisen during the skill mapping exercise. For instance, is there a difference between *creativity* and *being creative by approaching a problem from different angles*? Some skills are inherently difficult to map to the Delphi study list. *Ethics* and *role awareness*, skills appearing in the SLR but not in the last round of the Delphi study, are examples of this.

For the mapping process, we asked the assistance of academics instead of industry practitioners, because the source material originated from academia, and industry experts might introduce bias towards the Delphi skills. Some researchers who helped with the exercise reported running out of steam after connecting many skills, indicating cognitive strain as another possible limitation. To minimize this effect, we asked each researcher to only map either skills from the SLR or skills from the LO.

Next, the sample size of the conducted interviews is limited, and

all interviews were conducted in Dutch. The reason for this is twofold. First, the interviews served as a follow-up that supported, validated, and expanded upon the primary results of this chapter: the identified skill gaps. Since the primary source of our skill gap analysis is the Delphi study, which was conducted internationally across 11 different countries and 21 unique companies and universities, we felt comfortable enough to limit the interview breadth, but not the depth. Second, saturation assessment as shown in Table 6.2 indicates that the threshold has been reached after eight interviews.

Lastly, A possible threat to construct validity is the questions asked in the follow-up interviews in which there is an assumption in the question that assumes there is a possible lack of a certain skill. This could potentially be a leading question. However, as the goal of the interviews was to investigate the context of the identified skill gaps, we viewed this as the most natural starting point. As explained in Section 6.2, interviews were started with the design and purpose of the study, followed by a summary of the top three largest skill gaps that we identified.

We believe that these limitations have been circumvented as much as possible by adhering to the methodology elaborated in Section 6.2.

6.6 *Conclusion*

In this chapter, we aggregated and compared data about the non-technical skill gap in SE from previous chapters, with the help of eight senior educational researchers. By answering research question RQ1, we have identified the biggest non-technical skill gaps in the field of SE: *continuous learning*, *creativity*, and *solution-oriented thinking*.

By further exploring how the industry perceived the gaps for these three skills, using semi-structured interviews, we uncovered the context in which the skill gaps manifest themselves. Eight participants helped us to answer RQ2. This research yielded 26 sub-themes, grouped in six overarching themes: *stimulating*

continuous learning, *stimulating creativity*, *creative techniques*, *addressing the gap in education*, *skill requirements in industry*, and the *industry selection process*.

The research questions posed in this chapter also provide an answer to overarching RQ2 of this dissertation, which was:

PhD Research Question 2

How big is the gap between those requirements and software engineering education?

All three uncovered skill gaps are closely related, but lifelong or continuous learning is much more broad than creativity and creative problem-solving techniques. Furthermore, a quick search for related literature in SE education yields less results for creativity[6]. Therefore, to better focus this dissertation and to make a more original contribution, in the coming parts and chapters we will zoom in and shift our attention to creativity.

[6] This is also revealed in the second SLR of Chapter 8.

Part III

Creativity as a Non-technical Skill

In Part II, we discovered that the answer to overarching RQ2—How big is the skill gap—points towards three inherently linked gaps: *creativity*, *lifelong learning*, and *solution-oriented thinking*. In this part, we will zoom in on creativity.

Before being able to answer RQ3—how to improve education to reduce the gap—we first have to explore what exactly it means to be a creative programmer in context of SE. In Chapter 7, we discuss frequently used definitions in the field of cognitive psychology before making the jump to the domain of computing.

Next, in Chapter 8, we present a second SLR that identifies teaching trends of creativity in SE education (Groeneveld, Becker, and Vennekens, 2022a). These trends will help us better focus our own work.

Then, professionals' perceptions of creativity and students' perceptions of creativity are discussed and compared in Chapter 9 (Groeneveld, Luyten, Vennekens, and Aerts, 2021a) and Chapter 10 (Groeneveld, Luyten, Vennekens, and Aerts, in press) respectively.

Finally, a proposition to measure creativity in SE education is made in Chapter 11 (Groeneveld, Van den Broeck, Vennekens, and Aerts, 2022c). The chapters of this part will ultimately help providing the tools and context needed to answer RQ3 in Part IV, where multiple approaches to amplifying creative skills of SE students will be revealed.

7 Defining Creativity

Pinpointing characteristics of creativity is no easy task. Even though cognitive creativity and creative behavior are well-researched in the field of psychology (Petkus Jr, 1996; Simonton, 2000), it is important to note that there are dozens of definitions of the term "creativity" in literature, only adding to the confusion. Furthermore, in context of software development, creativity is severely underexposed. As Amin, Basri, Hassan, and Rehman (2017) concluded in their SLR:

> *The research work on creativity in SE is scattered and scarce.*

In this chapter, we will provide an overview of the current trends in researching creativity, starting with the field of cognitive psychology, briefly touching upon design, and finally investigating how this can be translated into SE.

7.1 The psychology of creativity

In the field of cognitive psychology, countless definitions have been proposed in literature, up to the point that it would be fruitless to try and list them all. Instead, we will sketch the evolution of the most important trends.

Psychological creativity research would not gain traction until the early sixties and seventies—it is still a relatively new field. Creativity was usually equaled to *divergent thinking*: coming up with multiple ways to implement something. When Torrance (1972) invented the

Torrance Test of Creative Thinking (TTCT) to assess creativity, the tool was rapidly adopted to assess both students' and job candidates' creative thinking capabilities[1]. Figure 7.1 illustrates a typical TTCT process.

[1] We will discuss assessing creativity at length in Chapter 11.



Figure 7.1: An illustration of a TTCT test: come up with as many drawings as possible using the starting shape. The more original pieces compared to others, the more creative that person is deemed. Suffice to say, this does not cover all aspects of creative thinking.

Creativity, however, is not limited to merely divergent thinking. Researchers Kaufman and Sternberg (2007) thus came up with a broader way to define creativity. Their then accepted definition is as follows: an idea is creative when:

1. it is considered novel and original;
2. it is of high quality; and
3. it is relevant to the task at hand.

This essentialists' take on creativity comes with many drawbacks. Note that Kaufman and Sternberg focused on idea generation instead of implementation, just like the TTCT does, without taking context into account. Furthermore, who will be the judge of novelty and originality? Kaufman later admitted to formulating a perhaps incomplete definition and also wonders: *who decides what is creative?* (Kaufman and Baer, 2012).

Since then, creativity research is making a gradual shift towards a more systemic approach that takes into account contextual parameters. Over twenty well-respected psychology scholars recently co-authored a manifesto to advance creativity theory and research (Glaveanu, Hanchett Hanson, Baer, Barbot, Clapp, Corazza, Hennessey, Kaufman, Lebuda, Lubart, et al., 2020), in which a new direction is outlined: creativity is a *social verdict*. The manifesto discusses the following statements:

- *Creativity is, at once, a psychological, social, and material (physical and embodied) phenomenon.*
- *Creativity is culturally mediated action.*
- *Creative action is, at all times, relational.*
- *Creativity is meaningful.*
- *Creativity is fundamental for society.*
- *Creativity is dynamic in both its meaning and practice.*
- *Creativity is situated but its expression displays both similarities and differences across situations and across domains.*
- *Creativity needs specification.*

- *Creativity research needs to consider power dynamics both within our analyses and as a field of study.*
- *The field of creativity studies needs both quantitative and qualitative methodologies with strong theoretical grounding.*
- *Old literature should be revisited and not abandoned.*
- *Creativity researchers have a social responsibility.*

Many statements correct or complement Kaufman and Sternberg's (2007) definition. For example, *creativity is meaningful*: creative outcomes are not only new and appropriate for a certain task (point 3 of the earlier definition); they can give meaning and even joy to our existence.

Modelling creativity

Theories of creativity are usually accompanied by models to identify a variety of factors that are influenced by different levels of creativity, of which the oldest well-known is Mooney's (1963) 4P model, representing *People* (or *Person*), *Process*, *Product*, and *Place* (or *Press*). The person is at the center of any creative endeavor, leveraging their environment (the place) and using their abilities in the process to ultimately create a creative product.

Amabile (1988) approaches creativity from another perspective: creativity, according to Amabile, is the result of *Expertise*, *Creative Thinking Skills*, and *Motivation*, as seen in Figure 7.2. This can be at individual level, at team level, or at organizational level. This model of creativity neatly matches Amabile's earlier work, the Consensual Assessment Technique (CAT) tool, that can be used to measure the creativity by enlisting a panel of judges (Amabile, 1982). Note that the CAT is quite remarkable as it was designed in 1982 and can still be employed since it adheres to the evolved definition of creativity as a social verdict[2].

Amabile's CAT is not without its shortcomings. Cseh and Jeffries (2019) critically evaluated the technique and call "the gold standard" of creativity assessment incomplete, as CAT methodologies often lack clarification and evidence. In order to harmonize the method while simultaneously preserving the flexibility of CAT, consistency and transparency should be

[2] As will be revealed in Part IV, we have applied this technique at multiple occasions. A detailed description of the CAT methodology will be revealed there.

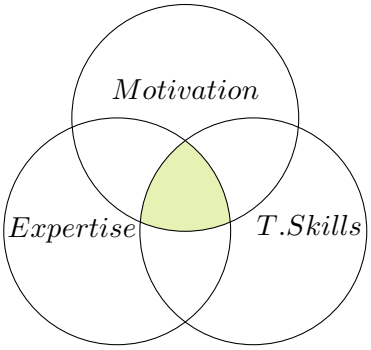


Figure 7.2: Amabile's Componential Model of Creativity, with at the intersection of the three components the emergence of creativity.

emphasized when publishing results. Reliance on judges inherently makes the evaluation of creativity subjective—which is exactly what the recent manifesto on creativity implies. When meticulously describing the approach, this does not have to be a disadvantage.

Still, plenty of guidelines for execution of CAT do exist. For instance, Baer and McKool (2009) suggest the following:

- Enlist on average 10 experts from a diverse background: try to on-board people from the industry.
- Rate creativity independently, but within same item pool, making the value relativistic, not absolute.
- Do not ask for experts for a "why" as to not influence the results.
- There is no need to rate in fractions (e.g. `1.0` to `5.0`). Although fractions are frequently used in grading, Baer and McKool (2009) suggest that ratings from `1` to `5` or `10` suffice.
- ...

Other lesser known componential models of creativity build upon that of Amabile and Mooney. For example, Urban's (2003) has 6 domains instead of 3 or 4: *divergent thinking & doing*, *general knowledge & thinking-base*, *specific knowledge base & specific skills*, *focusing & task commitment*, *motives & motivation*, and *openness & tolerance of ambiguity*.

Urban's domains suggest that personality traits—as listed in the popular Big Five personality model of Costa Jr and McCrae (2008) (e.g. *openness & tolerance of ambiguity*)—are related to creativity. Plenty of existing older and more recent work suggests this link as

well (Abdullah, Rozeyta, and Panatik, 2016; Barron and Harrington, 1981).

In an attempt to synthesize different models and frameworks that study creativity, Kuo (2011) listed, reviewed, and evaluated the most commonly used ones. They also conclude that there is a clear shift from modelling isolated traits to a more systematic approach. In other words, one should study creativity within its social context. This was already remarked by Csikszentmihalyi (2014):

> *Creativity is not an attribute of individuals but of social systems making judgments about individual(s).*

7.2 *Creativity and design*

Gero (2000) describes creative design by comparing it to innovative and routine design. He defines routine design in computational terms as an activity which occurs when all necessary knowledge is a priori available: a pre-established procedure can be followed to come to a design solution. In contrast, he defines non-routine design by two subgroups: *innovative* and *creative design*. In *innovative design*, the value of the variables directing the procedure to establish an outcome are placed outside the intended range. This leads to design outcomes that are new but still belong to the same class as their routine progenitors. In *creative design*, one or more new variables are introduced in the process leading to an all together new class of design outcomes.

Routine design lies firmly within the space of all possible designs as defined by the a priori decisions, while the space of true creative design lies outside of it. Figure 7.3 visualizes Gero's possible design spaces, which are reminiscent of earlier classification attempts of creativity by defining the following levels:

1. *LittleC* or everyday creativity. This is personal creativity: doing something original you have not done before. For instance, cross-compiling a C++11 game of life implementation to the Game Boy Advance. On a personal level, this could be compared to Gero's *innovative design*.

2. *BigC* or eminent creativity. Doing something original nobody has done before, resulting in Gero's radical *creative design*. For instance, porting Ruby 3 to run on an early nineties 80486 machine under MS-DOS 6.22.

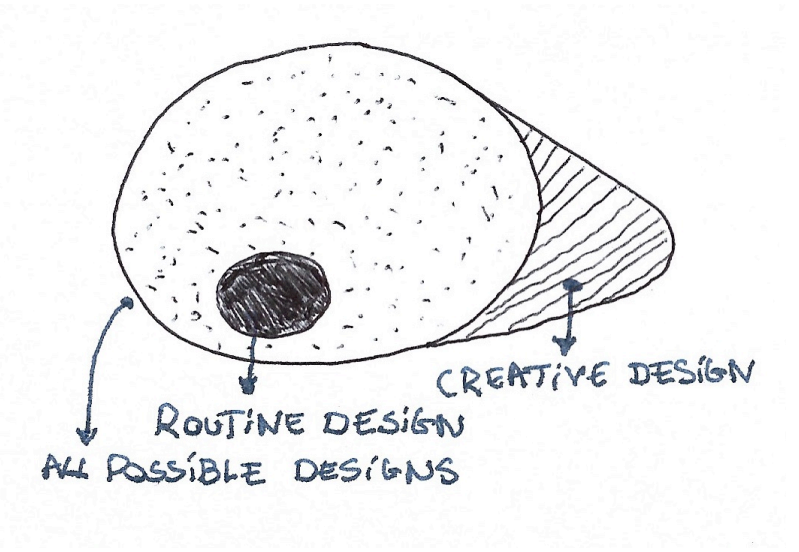


Figure 7.3: Gero's computational models of innovative and creative design processes, depicting all possible designs (the dotted circle), routine design within it, and radical creative designs extending it.

However, creativity researches have recently criticized this categorization: *LittleC* is sometimes portrayed as too mundane and bland, while the greatness of *BigC* might have creators succumb to the pressure (Kaufman and Beghetto, 2009). Beghetto and Kaufman (2007) advocate for adding *MiniC* and *ProC* levels in-between, offering a more fine-grained division of creative effort, while Runco (2014a) completely dismisses the distinction between little and big, proclaiming that creativity is not categorical. Others develop their own version in response, calling it *H-Creativity*—historic: does the invention impact the history books?—and *P-Creativity* or personal creativity (Boden, 1998).

Some researchers, such as Csikszentmihalyi (1997), interview *BigC* creative geniuses to extract practices for everyday creators, while others claim this gives a distorted picture and suggest we should investigate everyday *LittleC* creative efforts of professionals instead.

7.3 *Creativity and engineering*

REQUIREMENTS ENGINEERING IS ONE of the more popular software-related research areas where creativity has been studied. In an effort to better understand problem solving in requirements engineering, Cybulski, Nguyen, Thanasankit, and Lichtenstein (2003) discussed creativity with practitioners using focus groups. According to the authors:

> *Requirements engineering is well-recognized as a creative problem-solving activity by the systems development community.*

The framework utilized to guide the focus groups is reminiscent of Csikszentmihalyi's (2014) three-dimensional systems view of creativity: *context* (individual and social dimensions), *outcome* (the development of creative information systems), and *process* (supporting the creative characteristics of problem solving) form the main elements of creative problem solving. However, requirements engineering is not software engineering, and therefore, the findings grouped in these three dimensions might not be readily applicable in context of this work.

Bobkowska (2019) explored seven creativity techniques in SE using a specific training-application-feedback cycle. Since creativity research has produced more than a hundred different techniques, the question becomes whether or not any of these techniques are applicable to the field of SE. The following techniques were explored:

1. *naive questions*—discover hidden assumptions and implicit knowledge;
2. *reverse brainstorming*—first express criticism, then motivate to improve;
3. *Lunette*—look at the problem at different levels of abstraction;
4. *Chinese dictionary*—a technique to create atypical classifications;
5. *What if…*—search for hidden sequences of consequences;
6. *I could be more creative if…*—understand personal obstacles; and
7. *Let's invite him/her*—use creativity patterns of experts in creativity.

Participants left the experiment with an increased appreciation for creativity techniques, claiming that a mix of these techniques is likely to be useful in practice.

Even though cognitive creativity and creative behavior are well-researched in the field of psychology, it has proven difficult to position the exact role of creativity and its impact in computing education. For instance, some studies consider it to be related to computational thinking (Peteranetz, Wang, Shell, Flanigan, and Soh, 2018), while others view creative behavior as more akin to divergent thinking and still cling on to the TTCT technique (Salgian, Nakra, Ault, and Wang, 2013).

Besides various opinions on the building blocks of creativity, the discussion of whether or not creativity is domain-specific is still ongoing (Baer, 2010). If creative skills are domain-general, they are transferable from one domain to another: if you are creative in design, you could also be creative in SE. However, if creative skills are mainly domain-specific, they will be much harder to transfer. According to Nguyen and Shanks (2009), this debate is very relevant to creativity education.

Furthermore, existing studies attempting to understand creativity in the context of requirements engineering (Nguyen and Shanks, 2009) or computing in general (Veale, Gervás, and Pease, 2006) identify different contexts where different kinds of creativity seem to manifest: by yourself, in creative teams, or on socio-organizational levels.

7.4 *Conclusion*

UNFORTUNATELY, WE HAVE NOT come across a study within SE education that recognizes the evolution towards a more systemic approach as mentioned in the recent creativity manifesto from Section 7.1. Chapter 8 further explores which creativity measurement techniques are prevalent in SE research. For now, we have to agree with Amin et al.'s (2017) remark that opened this chapter: the research work on creativity in SE is indeed scattered and scarce.

SE Creativity in context of this work

Since existing work on creativity in the field of cognitive psychology is very broad and underexposed in SE, we will initially approach the term creativity with an open mindset. Instead of starting from certain definitions or techniques and matching them to software development like Bobkowska (2019) did, our aim in the following chapters is to let the answers come from the interviewees instead of the literature, while keeping into account the evolution of the interpretation of creativity in psychology research as outlined the previous sections.

But first, in order to better understand existing creativity research in SE, Chapter 8 presents a second SLR that identifies teaching trends of creativity in SE education. Then, professionals' perceptions of creativity (Chapter 9) and students' perceptions of creativity (Chapter 10) are discussed and compared. Chapter 11 ends Part III with a suggestion to measure the creativity, specifically geared towards SE students.

When exploring and discussing creativity in SE, we limit the scope of this dissertation to the following properties.

First, we are primarily interested in the day-to-day problem-solving capabilities of creativity—so-called *LittleC*—instead of the eminent one-offs such as the invention of a completely new database system.

Next, while the end product can be used to measure creativity, we are also interested in the programming process that led to that creative software product.

Also, while personality traits might exercise influence on creative behavior, focusing exclusively on personality assessment strides too much from the SE path. Instead, we are more interested in creative potential and how to amplify that.

Lastly, our aim is to move forward creativity research in SE education by bringing attention to the systemic properties of creativity. A holistic approach was taking to develop our creativity model, as revealed in Chapter 9.

8 Teaching Trends: Creativity

Publication details

This chapter has been published in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education* as *How Creatively Are We Teaching and Assessing Creativity in Computing Education?: A Systematic Literature Review* (Groeneveld et al., 2022a).
Its main contributions are identifying eight major themes in which creativity is employed in computing education research and by highlighting the interdisciplinary aspects of creativity research applicable in computing contexts. Additionally, it provides practical guidance and implications for educators in leveraging creativity in the classroom.

Part I and Part II show that creative skills play an important role in tackling difficult programming problems. As discussed in Chapter 7, creativity has been extensively researched in the field of cognitive psychology, and many of these methods can be adapted for application in computing education.

Creativity is often discussed in computing education research, but all too often not in any substantial way (Luxton-Reilly, Simon, Albluwi, Becker, Giannakos, Kumar, Ott, Paterson, Scott, Sheard, and Szabo, 2018), and a recent global study of introductory programming syllabi revealed no significant mentions of creativity

at all (Becker and Fitzpatrick, 2019c). When looking at curricula guidelines, such as the latest version of the latest version of the ACM/IEEE Computing Curricula (CC2020) (The IEEE CS/ACM Joint Task Force on Computing Curricula, 2020), there is no mention of creativity in a meaningful way apart from stating that it should be applied and developed, only mentioning `creativ*` once in the 93 page main report, and another 13 times in the appendices, most often in very specific contexts (for example, in a discussion on computing competencies in a specific country). Nonetheless, CC2020 states that:

> *A lack of diversity limits creativity and productivity* (*The IEEE CS/ACM Joint Task Force on Computing Curricula, 2020, p 15*).

Our initial pilot searches yielded many course experience reports published in venues sponsored by Special Interest Group in Computer Science Education (SIGCSE) which expressed an interest in creativity by mentioning the term. This suggests that the teaching of creativity is a topic of interest to the community. It seems that when creativity is mentioned, it is mentioned positively and with importance, yet, directly discussing or addressing creativity remains a niche activity.

To support research on creativity in computing education, this chapter presents a SLR of the teaching and assessment of creativity in the context of computing in higher education. We review current literature to explore which creative theories from cognitive psychology have crossed over into computing education research, how courses explicitly enhance the creativity of students, and how this is measured. Furthermore, we also summarize the approaches taken by the authors of the reviewed papers to implement creativity in computing education. We hope this is useful for researchers and educators who are seeking practical information on how to foster creativity in their own work.

We first describe the conducted review process and limitations before presenting and discussing our results including discussing theories that are notably absent absent from the context of computing education. We also summarize important approaches and results from these studies which use creativity as a theoretical basis, and highlights pedagogical implications of these studies.

8.1 The review process

This SLR is driven by the following research questions:

Chapter Research Questions

1. *What is the landscape of research on explicitly teaching creativity, in the context of computing in higher education?*
2. *What is the landscape of research on assessing creativity, in the context of computing in higher education?*

We followed the the SLR guidelines from Kitchenham and Charters (2007) and adapted the methodology of Prather, Becker, Craig, Denny, Loksa, and Margulieux (2020), a 2020 International Computing Education Research (ICER) SLR of metacognition and self-regulation in programming. Articles published in SIGCSE-sponsored venues and Transactions On Computing Education (TOCE) were collected by searching the ACM Digital Library. We also included articles from Taylor & Francis Computer Science Education (CSE). We searched abstracts for the string `creativity`. The Digital Library also automatically included variations of the term (e.g. `creative`). After initial exploratory searches, we decided against querying the full-text as it produced too many irrelevant results, often returning hits in bibliographies. We found that "creative" and variants are particularly common words, most often found in papers that do not focus on creativity. Our final search was conducted on July 19, 2021 and yielded 301 results.

An initial scan of the papers revealed that few papers clearly define or explain the terms creativity/creative. To gain a better understanding of the depth to which these papers explored creativity, we counted the number of times each publication contained the partial word `creativ`. Papers with fewer than five occurrences were removed. All of these were found to be extremely shallow in terms of the role of creativity. We also removed papers with fewer than three pages. Our SLR covers papers published from year 1994 to 2021 in the following venues and/or journals:

TOCE, SIGCSE, ITiCSE, ICER, ACE, WCCCE, and CSE. The full data set, including excluded papers, can be found at https://doi.org/10.5281/zenodo.5752559.

In summary, the following papers were excluded from our results[1]:

- 77 papers that were not about creativity in CSEd;
- 60 papers that did not explicitly target higher education (e.g., targeting K-12 or high school instead), and;
- 117 papers that were not research articles (e.g. posters, abstracts, keynotes), papers with less than three pages, or contained less than five occurrences of `creativ*` in the full-text.

[1] We have intentionally not limited our search to software engineering, but instead considered the broader computing perspective, as the teaching of creativity within that context is already considered scarce.

After this we further categorized the papers into three different classifications (depth, passing, and peripheral) in a manner similar to Prather et al. (2020). This helped guide our interpretation of the remaining papers. All 412 references of the depth papers were used to perform a snowball search in accordance with the aforementioned guidelines, of which 11 met our acceptance criteria, with only 3 as depth papers. The snowball results are included in Table 8.1.

Category	#	Descriptor
passing	23	the terms are mentioned in passing only, and not explored to any thoroughness
peripheral	17	the terms are discussed more thoroughly, but are not used to support the methodology of the paper or interpret results
depth	18	the paper uses creativity as a central theoretical basis of the study and measures/interprets the results using these theories

Table 8.1: Categories and descriptors for paper classification, from Prather et al. (2020), including counts for each category (58 total).

The first two authors independently categorized all 58 papers, after which an initial inter-rater reliability (IRR) was calculated. Following McAlister et al. (2017), we calculated an IRR of 0.83. Next, a round of discussion reconciled all differences, including possible different interpretations of the classifications. When both authors agreed upon a classification, we started investigating the papers marked as depth more thoroughly to help us move towards answering our research questions.

Limitations

Our primary audience is the SIGCSE community. Therefore, we focused our search on literature published in SIGCSE-sponsored venues, TOCE and CSE. A snowball search further extended the results into other disciplines.

Nonetheless, more sophisticated search approaches are possible. There are certainly papers on creativity in computing education published elsewhere that are not included in our results. For instance, the work of Oihus, Surovek, and Jensen (2013) on developing student creativity through competition, published in an IEEE venue, would have been classified as a depth paper.

However, an exploration of IEEE literature did not lead us to believe that there are many such papers. Furthermore, papers that do not contain our search term were omitted. Although it is unlikely that one could write a paper on creativity without mentioning the word itself, we are aware of linked concepts that are expressed differently by some. For instance, papers on Computational Creativity Exercises by Peteranetz, Soh, and Ingraham (2019) are included, while Computational Thinking Exercises are not (Lu and Fletcher, 2009) unless creativity is mentioned elsewhere in the paper.

Since our primary goal is to investigate the landscape of research on explicitly teaching or assessing creativity in computing education, we do not expect that these possible oversights would diminish the contribution of our work. However this work should be viewed as one piece to a complex picture of creativity in computing education and should be interpreted as such.

8.2 Results & discussion

As shown in Figure 8.1, the body of research on creativity in computing education has increased in the last ten years. The earliest included work dates back to 1994. The greatest number of in-depth creativity-related papers for a given year was in 2018 (6). Yet, compared to other SLR studies (Prather et al., 2020), the volume of papers we encountered was surprisingly low. This

agrees with Amin et al. (2017) who concluded in their 2017 SLR on creativity in SE that the research work on creativity is severely underexposed. Our results indicate the same for creativity in computing education. As stated in Section 8.1, we categorized the remaining papers as being either *passing*, *peripheral* or *depth*.

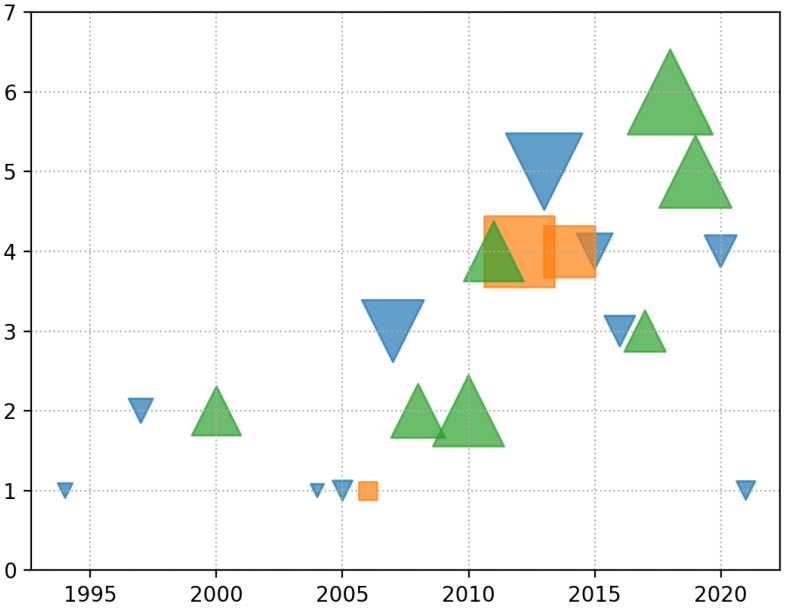


Figure 8.1: Scatter plot of papers by year, categorized by: depth (△), peripheral (□), passing (▽). Symbol size is proportional to the number of instances of *'creativ'*.

Passing & Peripheral

Nearly half of the papers (23 of 58, or 40%) were categorized as *passing*. Many studies mention creativity (more than once) but without thoroughly discussing it in any significant capacity. For example, in a paper that describes a CS1 course designed to address interests of women (Rich, Perry, and Guzdial, 2004), the authors introduced "creative freedom" to counter the reputation of CS as being boring. By interpreting a few interviews with students, the authors claim to demonstrate that the course was "creative", yet they provide no formal definition of creativity, nor a theoretical basis to measure and/or interpret the results. Most *passing* papers are in fact experience reports that also happen to state the course

was "more creative" than prior or alternative courses, or similar. For such a paper to be categorized as *depth*, these claims would need to be backed up with data and/or theory as we are interested in reports on explicitly teaching creativity (RQ1), or assessing creativity (RQ2), not in reports that claim either without including evidence or theoretical grounding.

In total, 17 papers (29%) were classified as *peripheral*. These papers explain the context of creativity in more detail than the *passing* papers, but without it being a central focus of the study. For instance, in a 2020 paper on service learning (Vennekens, 2020), the authors aimed to increase the skill set of students by focusing on creativity and empathy. Instead of simply mentioning it in passing, the authors defined their understanding of creativity and empathy. In another example, creativity was "promoted" using a computer science design studio (Cennamo, Douglas, Vernon, Brandt, Scott, Reimer, and McGrath, 2011). This paper also defined creativity and described a methodology to facilitate creative thought. However, the results did not feature creativity to the extent of the *depth* papers.

We initially attempted to identify themes for the peripheral papers. However, only 5 out of 17 papers were easily agreed. After a brief discussion, we decided against further exploring these papers. We believe that the majority of peripheral papers are not primarily *about creativity*, despite having a more than passing facet (e.g. a separate section) on creativity. This proved to be not focused enough for us to identify coherent themes in the majority of these papers.

Depth: Themes & Theories of Creativity

In the following subsections, the 18 papers classified as *depth* are categorized by major theme, the creativity theories they employed, and their measurements of creativity. In Section 8.3, we highlight pedagogical implications of these studies. Our discussion is structured around common themes and theories of creativity, followed by a short overview of notable creativity theories that are underused in the field of computing education. Theories on which assessment tools are based are discussed in the following section.

Emerging themes

We found eight distinct themes of creativity in depth papers, as shown in Table 8.2. The most prevalent theme was *"Teaching CS0/1/2"*, where the main focus is on introductory sequence programming classes. These papers are primarily experience reports that use creativity to enhance the teaching and learning of a specific course. The next most prevalent theme was *"Computational Creativity"* papers which blend computational thinking and creativity in exercises to "enhance students' computational thinking skills, creative competencies, and self-efficacy" (Peteranetz et al., 2019) or similar.

Theme	Papers
Teaching CS0/1/2	Miller, Soh, Chiriacescu, Ingraham, Shell, and Hazley (2014)
	Jones, Kisthardt, and Cooper (2011); VanDeGrift (2015)
	Xu, Wolz, Kumar, and Greenberg (2018)
Computational Creativity	Peteranetz et al. (2018,1)
	Shell, Soh, Flanigan, Peteranetz, and Ingraham (2017)
Environment Creation	Isomöttönen, Daniels, Cajander, Pears, and Mcdermott (2019)
	Göttel and Schild (2011); Krusche, Dzvonyar, Xu, and Bruegge (2018)
Explicitly Teaching Creativity	Apiola, Lattu, and Pasanen (2012); Salgian et al. (2013); Shneiderman (2000)
Intrinsic Motivation	Apiola, Lattu, and Pasanen (2010); Knobelsdorf and Romeike (2008)
Problem Identification	Brookes (2018)
Collaboration/Teamwork	Yang and Cheng (2010)
Curriculum Design	Romeike (2007)

Table 8.2: *Depth* papers: major themes

It should be noted that these computational creativity papers all share a common co-author (Peteranetz). Papers categorized with the third most prevalent theme, *"Environment Creation"*, focus on creating the right physical *"creative environment"* inside and outside the classroom. For instance, in a game design course, students were offered unique meeting rooms that offered playful and creative elements (Göttel and Schild, 2011). Papers within the theme *"Explicitly Teaching Creativity"* focused on creativity more than anything else, and do not use a proxy for tapping into creativity. In these papers, computers just happen to be a vehicle for exploring creativity, or being creative. Interestingly, we only found 3 out of 18 depth papers to be about explicitly teaching creativity. The fifth identified theme was *"Intrinsic Motivation"*,

which contained papers that use creativity to increase student motivation, engagement, en enthusiasm. *"Problem Identification"* papers used creative ways to identify the problem instead of applying creativity to a solution, while *"Collaboration"* papers use creativity to emphasize teamwork and presentation skills. Lastly, *"Curriculum Design"* papers focus on introducing creativity beyond the borders of a single course.

Emerging theories

As discussed in Chapter 7, Amabile's (1983) Componential Model of Creativity defines the requirements for creativity as the right cognitive processes and working styles (the ability to know good from bad ideas), intrinsic motivation (be willing to pull ideas as part of genuine interest), and domain-relevant skills (the information database to pull from). According to Amabile, a person's best creativity can be found where these three components overlap the most. These three components also occur in the creative model of Csikszentmihalyi (1997), as the domain (knowledge, tools), field (community of practice), and person (individual practitioner), with emphasis on social context. In one paper that applies these three components, a learning environment is created to promote what the authors call competence, autonomy, relatedness, cognitive processes and working styles, domain-relevant skills, and constructionism (Apiola et al., 2012).

The debate in the literature of the domain-specificity of creativity has also been noted by the authors of papers we reviewed. Salgian et al. (2013) argue that not only is creativity domain-specific, as Baer (2010) suggested, but there are also multiple types of creativity, of which the authors identified the following: creativity of design, of problem solving, of knowledge acquisition, and of self-definition.

The 4P creativity model, originally conceived by Rhodes and frequently mentioned by Runco (2014b), describes four major dimensions of creativity: (1) person (personality, attitudes, self-concept, behavior), (2) process (motivation, perception, thinking, learning, communicating), (3) press/place (the relationship with the environment), and (4) product (the idea that is transformed into a tangible form). Knobelsdorf and Romeike

(2008) integrated these creativity dimensions into their view of a more creative curriculum pathway to CS, with Romeike (2007) looking beyond the borders of a single course.

Notably absent theories

In this section we discuss a selection of theories from cognitive psychology that were notably absent from the papers in our review. In 1991, Sternberg and Lubart (1991) suggested specific assets of six resources to contribute to creativity: (1) intellectual processes, (2) knowledge, (3) intellectual styles, (4) personality, (5) motivation, and (6) environmental context. These aspects are also found in other theories, such as Amabile's three-componential model. However, there is more emphasis on personality and intellectual style in this case. Depending on the domain, Sternberg & Lubart propose the strategy of *"Buying low and selling high"*. Buying low links risk taking to creative performance, while selling high links the ability to present one's work at the right time. Failing creativity could also be the result of holding on to an idea so long that it becomes obsolete. Thus, students should be introduced to the idea of risk-taking and letting go of projects when necessary.

Urban (2003)'s 2003 componential creativity is a continuation of the work of Amabile & Csikszentmihalyi. In this model, five rather than three components are identified, each intersecting with three dimensions of environment on individual, group or local, and societal level. In this sense, Urban's model is a combination of that of Amabile, Csikszentmihalyi, and the "place" dimension of Rhodes' 4P model. As Urban noted, we should keep the social context in mind when teaching students creativity. Individual creativity is a wholly different beast than creative techniques in groups.

Lastly, Unsworth (2001) "unpacked" creativity in 2001 with the Matrix of Creativity Types. This model advocates against the assumed homogeneity of creativity by introducing four distinctive creative types: responsive, expected, contributory, and proactive. According to Unsworth, creativity is influenced by the dimension of *"why?"* (the driver behind the engagement - comparable to Amabile's intrinsic motivation) and *"what?"* (the degree of

problem-finding needed—also found in the research of Csikszentmihalyi). Every combination of why (external/internal) and what (closed/open) represent one of the four creative types. For instance, Unsworth states that responsive creativity:

> [...] *is the most prevalent form of creativity studied. The strong historical focus on creativity testing has meant that many studies present participants with a problem (the creativity test) and place external demands on them to engage in creativity. Thus, models derived from these studies may be relevant only to responsive creativity* (*Unsworth, 2001, p 291*).

Furthermore, in order to amplify creativity, engaging students and asking open questions is more important than serving them the obligatory closed problem.

Measuring Creativity

There are myriad ways to measure the creativity of students, from gauging creative potential to dissecting creative problem-solving techniques. Many assessment tools from the field of cognitive psychology exist, such as the well-known TTCT (Torrance, 1972). However, only a few authors effectively use these instruments: most researchers simply resort to interpreting surveys or analyzing course grades. Table 8.3 contains a summary of the measurement instruments we uncovered.

Instrument	Papers
Survey/Interview	Apiola et al. (2010,1); Brookes (2018); Göttel and Schild (2011) Isomöttönen et al. (2019); Jones et al. (2011); Krusche et al. (2018) Peteranetz et al. (2018); Salgian et al. (2013); VanDeGrift (2015)
Self-reported scales	Brookes (2018); Salgian et al. (2013); Yang and Cheng (2010)
Computational Creativity Exercises	Peteranetz et al. (2018,1); Shell et al. (2017)
Behavior observation	Göttel and Schild (2011); Krusche et al. (2018)
Course grading	Miller et al. (2014); Peteranetz et al. (2018)
Consensual Assessment Technique	Salgian et al. (2013)
Torrance Test (and variants)	Salgian et al. (2013)

Table 8.3: *Depth* papers: creativity measurement instruments

We discuss these in turn in the following subsections.

Surveys/interviews

Qualitative surveys and interviews are by far the most popular method to measure the creativity of students: 10 papers (56%) employed either a qualitative survey or a combination of survey and interview. After introducing changes in a course that might improve creative thinking, students are often interviewed or asked to fill in a survey, which usually contains open-ended questions. The answers are then coded and interpreted by researchers. However, using this method, it seems hard to quantify the amount of "creative gain".

Self-reported scales

Sometimes, surveys also include specific questions that self-report creativity on a scale (Salgian et al., 2013, Brookes (2018),Yang and Cheng (2010)). Rating one's own creativity is especially difficult if the definition of creativity is still a matter of debate. Therefore, Salgian et al. also asked students to define it, resulting in the vast majority replying "*to think outside the box*" (Salgian et al., 2013). However, this is likely a function of the study context and may not generalize.

Computational creativity exercises

We encountered three papers, all sharing a common co-author, that use the term "*computational creativity exercises*". In these, the link between computational thinking and creative thinking has been measured with the help of a researcher-developed instrument adapted from the Epstein Creative Competencies Inventory (Epstein, Schmidt, and Warfel, 2008). In one study, this was complemented with a computational thinking/CS knowledge test and a survey on self-efficacy (Peteranetz et al., 2018).

Behavior observation

In two papers, the authors observed student behavior during multiple creative assignments. For instance, Göttel and Schild (2011) took note of the way students utilize their environment to

increase their creativity. The authors created a special "creativity room" to enhance the creative flow and ultimately improve the design of the game assignment. However, the observations alone were not enough to draw conclusions, and a survey sent out to students supplemented the notes of the researchers.

Course grading

Some studies use course grades as only one of the many dimensions to measure creative progress (Peteranetz et al., 2018), while others use it as the primary instrument to determine impact on student achievement in the course (Miller et al., 2014). The latter study used what they call "*creative thinking exercises*" to increase performance and retention. Thus, creativity itself was not the measurement, but the means to improve grades.

Consensual Assessment Technique and Torrance Tests

Only one paper employed both Amabile's (1982) CAT and a variation of the TTCT (Salgian et al., 2013). The CAT technique is used as a performance measurement in which experts on a given subject rate a product, such as a CS1 project, on a ten-point scale. This gives a more accurate representation of the creative work of the student, compared to self-reporting creativity in surveys. The CAT assessment is a good choice for a holistic approach that implicitly includes all dimensions of creativity, since it assesses the end product. On the other hand, the TTCT tests primarily measure divergent thinking.

8.3 Approaches, tools, & pedagogical implications

This section summarizes papers classified as *depth*. First, approaches and tools used to enhance creativity are discussed, including examples. Next, future work deemed relevant is summarized. Our intention is to provide researchers and educators with practical information that can be integrated into their own work.

Introduce experimental learning

One approach to improve the motivation, and possibly the creativity of students is utilizing themes that are more appealing to students. For instance, open assignments on game design and using LEGO® Mindstorms in robotics courses (VanDeGrift, 2015, Apiola et al. (2012),Göttel and Schild (2011)). Apiola et al. state:

> *The robotics kit seemed to work well both as a trigger for motivation and as a platform to support experimental learning, enhancing students' creativity and working style.*

Open-ended project-based learning has been shown to be beneficial towards students' creativity. For instance, in searching for what the authors call "global employability", Isomöttönen et al. (2019) designed a course to include open resources (facilitating global awareness), self-reliance and decision making (facilitating a shared responsibility), and group work (facilitating communication skills and diversity). The open nature of a project also seems to trigger students' creativity to initiate the use of open source libraries.

Apiola et al. (2012) recommend future work to investigate why some students perform better and feel more comfortable in open learning systems compared to others. They suggest looking further at the interplay between creative problem-solving abilities and psychological characteristics, combined with phenomena of social interaction. We found no depth papers that employ personality or standardized tests to correlate with other creativity metrics.

Open assignments solve another issue prevalent in many curricula. According to Brookes (2018), creativity appears only in curricula to produce "creative solutions" to known problems, rather than identifying *new problems* to address. Thus, Brookes advocates for the inclusion of innovative dimensions of creativity that help organizations advance, compete, and differentiate themselves. Therein, open-ended design challenges should also focus on a problem-finding component. This would be strengthened in partnership with industry, as assignments then become relevant and valuable beyond being an assessment of knowledge.

Getting students out of the classroom

One of the major themes identified in Section 8.2 is *"Environment Creation"*: setting the right scene to amplify the learning experience of students. A common approach to this is to simply get students out of their usual comfort zone—literally. For instance this can be achieved by designing a special "creativity room" with easy access to tools such as whiteboards, trinkets, and props that stimulate the creative mind (Göttel and Schild, 2011). Another slightly less drastic approach is to bring students with different disciplinary backgrounds together, such as pairing computing and linguistics students. This not only generates interesting discussions that normally would not occur, but also invites students who would not consider studying programming to potentially identify an unrecognized talent (Jones et al., 2011).

Another way of getting students out of the classroom is to bring them in contact with industry, an approach often implemented in capstone projects (Krusche et al., 2018). It is worth noting that, although classified as a peripheral paper, service-learning has been observed to be effective in raising both creative and empathic skills (Vennekens, 2020). Vennekens also argues for getting students in direct contact with a target group outside of the borders of the university. Several authors remark that while much creative work might indeed happen outside of the classroom, reflecting on the learning activity, and focusing on creativity in the classroom is certainly a worthwhile endeavor.

Göttel and Schild (2011) also note that their creativity room strengthened the group bond and a sense of belonging, a subject that has been paid increased attention in computing education recently (Mooney and Becker, 2021). Although G"ottel & Schild did not formally measure this, it is an interesting starting point for future work. Also, they note that for larger student groups and projects, the use of agile methods such as Scrum could be used to more effectively manage work, leaving more time for the creative work itself. This suggests that reducing overhead might also allow for an increase in creativity.

Putting creativity 1st, programming 2nd

Jones et al. (2011) stated *"put creativity first, programming second"*. This approach was shown to positively affect the design and implementation of an introductory programming course, co-taught by English and Mathematics/CS staff. It was found that this attracted and retained more computing students, as the authors found that many students find programming (without the explicit creative part) to be a dry and tedious task, doable only by students gifted with strong analytical skills—a view that has been challenged (Becker, 2021). In addition, many peripheral and passing papers focusing on gender and diversity use creativity to broaden interest in computing (Bares, Manaris, McCauley, and Moore, 2019, doube2012gender). Future work could focus on exploring this further by providing a theoretical basis and meaningful evidence to support such claims.

Shell et al.'s (2017) approach to merge computational thinking with creative thinking into computational creativity exercises leads to increased student learning in courses across all years of a degree. They note that students appear to benefit from such exercises, whether they are personally motivated to do them or not. The current exercises are carried out individually. However, the authors plan to look into the importance of collaboration as one of the essential skills for effective problem solving.

The work of Xu et al. (2018) emphasizes creative self-expression as an approach for diverse student populations within a broad range of pedagogical frameworks. By first focusing on creativity rather than the theoretical aspects of computing, they found their approach to catalyze excitement and increase engagement, stating that (emphasis theirs):

> *Programming is essential to learning computer science, but the coding must be purposeful. In the curriculum presented here, that purposefulness comes from the emphasis on creative arts. This speaks to a need for reconsideration of}* ***how*** foundations of programming are introduced rather than ***depth of knowledge*** of those foundations (*Xu et al., 2018, p 172*).

Of course, creative expression is not the only *how* framework, but seemingly a very effective one. Furthermore, they note that

adapting the curriculum to meet those needs requires teacher professional development. A compelling question for the computing community is how to provide those resources in a way that not only engages students, but also those teaching them. This raises the question of how to creatively teach teachers creativity, before teachers can teach creativity effectively, to students.

8.4 Conclusion

We conducted a systematic literature review of 301 computing education papers focusing on creativity. These were reduced to 58 papers categorized as *passing*, *peripheral*, or *depth* according to a previously published methodology. We focused on the depth papers in terms of explicitly teaching creativity, assessing creativity, and the use of creativity theories. We discussed creativity theories that were absent from our results, and pedagogical implications of the approaches to creativity in computing education that were found. We would like to highlight the following approaches taken to foster student creativity:

- Introduce experimental learning. Experiments with open-ended project-based learning have been shown to be beneficial towards students' creativity.
- Get students out of the classroom. The environment in which the learning takes places plays a significant part in the creative potential of students.
- Put creativity first, programming second. This opens up computing to a more diverse student population.

This work can serve as a resource for the computing education community, helping researchers find relevant work regarding creativity. Additionally, we provide guidance and implications for leveraging creativity in the classroom.

In chapters 9 and 10, to complement these SLR findings, we further explore creativity in context of problem solving in SE from both the viewpoint of the professional and that of the student. In Chapter 11, we propose a creativity measurement tool and the applicability of it for computing students in higher education.

9 Professionals' Perceptions of Creativity

Publication details

This chapter has been published in the *2021 IEEE/ACM 43nd International Conference on Software Engineering: Software Engineering in Society* (*ICSE-SEIS*) as *Exploring the Role of Creativity in Software Engineering* (Groeneveld et al., 2021a). Its main contribution is clarifying the concept of creativity related to problem solving in SE. The paper also identifies which creative techniques are used by industry professsionals, how creativity can be recognized, the reasons for being creative, and what environment is needed to facilitate creative work.

Chapter 8 summarized the landscape of research on explicitly teaching creativity, and identified a sobering small number of papers, of which many hold on to outdated views of creativity, as clarified in Chapter 7. Also, in context of SE, we know thanks to Chapter 4 that creativity is an important non-technical skill, yet we do not know how exactly creativity is perceived by industry experts.

SE is said to be an outcome of human knowledge and creativity (Bjørnson and Dingsøyr, 2008). Therefore, we wonder:

Chapter Research Questions

1. *What exactly is the role of creativity in the world of SE?*
2. *What makes one software developer very creative and the other less so?*

In order to approach the research questions with an open mind, we invited 33 experts from the industry to discuss creativity, as explained in the following section.

9.1 The focus group procedure

We conducted four focus groups, collecting information from developers from different agile software development companies. A focus group is a small group of experts in a specific field, who brainstorm about a specific subject in guided open discussions, to gain a better understanding on the subject at hand (Onwuegbuzie et al., 2009). In this case, experts are experienced developers and the subject is creativity.

The focus group method was adapted for SE based on the work of Kontio et al. (2004), as they state:

> [*It*] *is best suited to obtain feedback on new concepts,* [...] *generating ideas.*

To simplify the organization and to increase the likely-hood of participating, each session was held at a separate company. Instead of using purposive sampling techniques to compose the focus groups, they were formed by using preexisting groups, namely colleagues. According to Fern and Fern (2001), this is not bias but rather an advantage, since it has the added advantage that participants feel comfortable as they know each other. Due to the COVID-19 pandemic, the last two sessions were held online.

The focus group process

Expert selection was done by limiting participation to people with a technical role that come into daily contact with source code, such

as *developers*, *programmers*, *software architects*, ... To further increase relevance of answers, experts were required to have at least six years of experience in the field, and should be intrinsically motivated to partake in the discussion. Participation was entirely voluntary. Note that, although the job title of many participating experts is *developer*, this does not mean that their job is limited to merely implementing pre-set requirements. The four agile software development companies involved define the role *developer* as someone who is involved in both analysis, design, implementation, support, and maintenance phases of a software project.

The conversations were facilitated by the first author, since he has more than 11 years of experience in the industry. Even if some researchers claim that familiarity with the topic will introduce bias (Fern and Fern, 2001), Kontio et al. (2004) state that:

> *In the field of SE, it is very important for the interviewer to have extensive knowledge of the theme of the interview.*

To reduce bias as much as possible, each session was audio-recorded, transcribed, and cross-validated by the second author, whose field of expertise is architecture rather than SE.

During two hours, open questions are used to elicit opinions, perceptions, and ideas, which further help us as researchers to determine the role of creativity in software development. In order not to influence the results, we chose to refrain from providing a clear definition of creativity. By not defining the concept ourselves, we try to find out what the industry might mean when they seek "creative" software engineers. The following questions were addressed:

1. *What is the most creative thing you ever did in your job?*
2. *How can you see when a colleague is being creative, or not?*
3. *What is the most creative project you ever worked on?*
4. *How could you measure how creative someone is?*
5. *Which creative techniques did you recently employ?*
6. *What is the most important reason to be creative?*
7. *Which environment do you need to be creative?*

As an ice-breaker, participants first had to complete the

sentences "*as a developer, I am creative when I ...*" (1) and "*as a developer, I am not creative when I ...*" (2). After this individual warm-up exercise, the facilitator gave a brief overview of Amabile's (1988) Componential Model (Expertise, Creative Thinking Skills, Motivation) and Mooney's (1963) 4P Model (Process, Product, People, Place) to increase the awareness of the multi-dimensional aspects of creativity in general. Next, the above questions were asked in random order, where participants were grouped in pairs to brainstorm for five minutes, followed by ten minutes of discussion in group.

Processing focus group data

Data was identified by concepts that "bubble up" and present themselves after multiple data gathering and reduction steps, as shown in Figure 9.1 and presented by Onwuegbuzie et al. (2009). First, focus group recordings were transcribed (1) by the first author. Then, the transcript and written notes were read multiple times to apply an open coding step (2), yielding 82 codes. For this, we followed the guidelines of Richards and Hemphill (2018). All transcripts and notes were analyzed by the first two authors simultaneously in order to identify patterns. Afterwards, notes were compared and cross-validated where needed. Then, codes were further categorized in an axial coding step (3), yielding 39 subthemes. Multiple revisions of mind maps were made individually and discussed in group to finally arrive at seven main themes (4) after merging results from each focus group session.

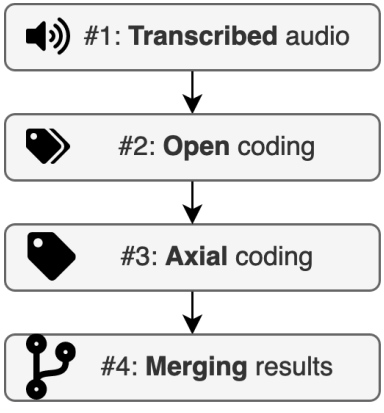


Figure 9.1: The focus group data processing steps.

After conducting the second focus group and merging results, the calculated thematic data saturation in accordance with Guest et al. (2020) was not yet reached (information threshold: 27%). After the fourth session, the threshold dropped below the pre-set 5%, indicating the end of the data collection phase. This effect is visible in Figure 9.2.

We have used an *emergent-systematic* focus group design (Onwuegbuzie et al., 2009), in which we organise multiple focus groups to assess if the themes that emerged from one group also emerge from other groups.

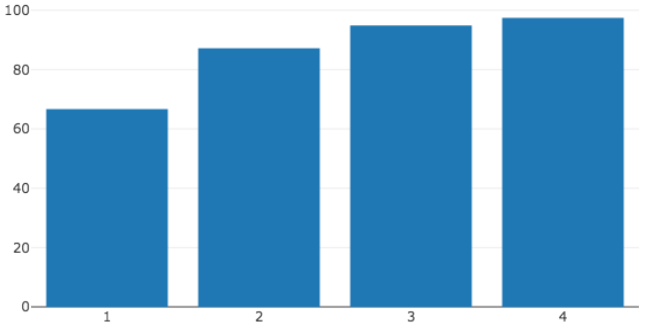


Figure 9.2: Data saturation: focus groups (x-axis) projected on the percentage of discovered codes (y-axis). Theme distribution rates are also discussed in the next section.

9.2 *Results & discussion*

TABLE 9.1 REPRESENTS FOCUS GROUP metadata. A total of 33 participants contributed to 399 minutes of transcripts[1], yielding a total of 52.329 words. The average number of participants for each session was 8, with an average duration of 100 minutes. Companies involved differed in size, from national to internationally renowned. This, combined with the participant selection process as explained in Section 9.1, led us to believe that respondents have accumulated sufficient experience to be able to contribute to this research.

[1] In accordance with the ICSE 2021 Open Science Policies, the transcripts of the focus groups are available upon request. The scripts used to analyze the codes and generate the data saturation graph Figure 9.2 are available on https://people.cs.kuleuven.be/~wouter.groeneveld/creafocus/.

Group	Duration	Participants	Words	Theme distr.
Total	399m	33	52.329	/
#1	105m	7	13.454	67%
#2	99m	14	12.296	87%
#3	100m	7	13.492	95%
#4	95m	5	13.087	100%

Table 9.1: Focus group metadata, including theme distribution rate to gauge data saturation.

The seven guidance questions resulted in a lot of similar answers. Therefore, we opted to group themes in subsets of these questions,

each of which is briefly discussed. The mind map in Figure 9.3 was created to provide feedback to respondents on the seven identified categories and 39 subthemes. It was received very positively and deemed complete: no new themes were added.

The full list of the 39 identified subthemes can be consulted in Table 9.2, with their occurrences in each conducted focus group. The number of emergent themes drop below the pre-set threshold of 5% after the fourth session, indicating thematic data saturation.

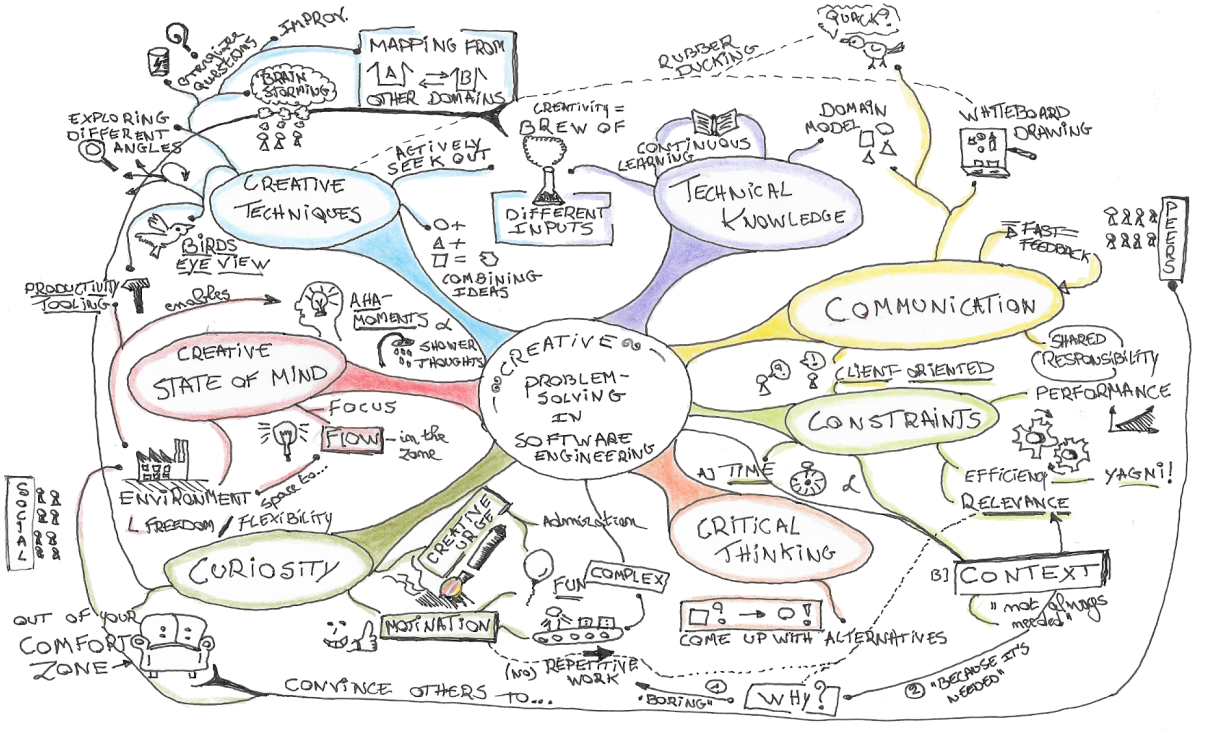


Figure 9.3: A mind map that summarizes identified themes on creative problem solving.

I am (not) creative when I ...

The warm-up exercise yielded surprisingly uniform results, where developers generally agreed what it means to be creative in their line of work. A few participants tried to define creativity:

When someone can come up with an elegant solution to a previously unsolved

Subthemes occurring in focus group:	1	2	3	4
Creative Techniques				
Employing a *birds eye view*		✓		
Exploring the problem from *different angles*	✓			
Asking *energizer questions* to bootstrap the creative process		✓		
Different forms of *brainstorming*	✓	✓	✓	✓
Mapping solutions from *outside the problem domain*	✓	✓		
Actively *seeking out new information*			✓	
Combining multiple ideas into a better one		✓	✓	
Technical Knowledge				
Creativity as the *'brew' of different inputs*			✓	
Continuous learning	✓			
Using *domain modeling* to gain new insights	✓			
Communication				
Using the *'rubber duck'* when stuck	✓	✓	✓	
Visualizing the problem on a *whiteboard*	✓	✓	✓	✓
Getting *fast feedback*	✓	✓	✓	✓
Sharing responsibility with all team members	✓			
Client-oriented designing		✓		
Working closely with *peers*	✓			
Constraints				
Putting emphasis on *efficiency*				✓
Putting emphasis on *relevance*		✓		✓
Putting emphasis on *performance*			✓	
'YAGNI': *You Aint Gonna Need It*	✓			
Time-constraints	✓	✓	✓	✓
Relevance of *context*	✓	✓	✓	✓
Critical Thinking				
coming up with *alternatives*	✓		✓	
Curiosity				
Start with *'Why?'*	✓			
The dangers and joy of *repetitive work*	✓	✓	✓	✓
Complexity as a motivational factor		✓		
Programming *'for fun'*!	✓	✓		
Admiration for creative work of others	✓			
Giving in to that *creative urge*	✓		✓	
Motivation in general	✓	✓	✓	✓
Convincing others work creatively		✓		
Getting out of that *comfort zone*	✓	✓	✓	
Creative problem solving as a *necessity*		✓		
Creative State of Mind				
The *social nature* of software development	✓	✓		
An environment that facilitates *freedom*	✓	✓	✓	✓
An environment that facilitates *flexibility*	✓			✓
Being in a *creative 'flow'*	✓	✓	✓	✓
'Aha' moments and shower thoughts	✓		✓	
Employing different *productivity tools*	✓			
Total percentage of *found* themes in each group	69.23%	56.41%	46.15%	30.77%
Total percentage of *new* themes in each group	66.67%	20.51%	7.69%	2.56%

Table 9.2: The identified themes and subthemes on creativity.

non-trivial problem.

However, after a brief discussion, it was criticized: it should be unsolved by the developer, not by anyone, making creativity something inherently *personal*. Also, it does not need to be elegant: coming up with an inventive quick but ugly hack that works is also deemed as creative. The problem at hand should be *complex*; otherwise you are simply applying previous knowledge to finish a simple task (cf. Gero's (2000) routine design of Chapter 7). Everyone agreed that they are creative when they model or visualize problems, split problems into smaller parts, do complex refactoring work, and brainstorm to approach a problem from another angle.

When are software engineers not so creative? When they blindly take over code without asking anything, when no alternatives are being devised, when tightly defined tasks such as simple TODO lists are being ticked off, when generating boilerplate code, and when *repetition* kicks in:

If I just try it until it works, such as fixing dependencies.

Engaging in *symptom relief* with quick-fixes, rather than addressing the root cause, has likewise been mentioned. Participants also talked about the dangers of *tunnel vision*, where one is not open to alternatives, consciously or not. Some developers claim that working on legacy code is not creative, while others state that legacy code comes with more constraints that require more creative work compared to maintaining modern projects.

Lastly, interrupts are frequently mentioned as distinctively breaking the creative flow. It has been proven before that "deep work", being in the flow of work, is of great help in solving problems (Newport, 2016).

Requirements to be creative

In the discussion we identified several requirements or conditions that are necessary for creativity to take place or to improve it. They range from personal traits, such as creative skills and sufficient

technical knowledge, to requirements related to the physical, cognitive, and emotional context of work.

Personal: technical knowledge and creative skills

There seems to be a thin line between a hack (creative) and fiddling (ignorance). This means that *technical knowledge* is important, as also denoted by Amabile's (1988) Model. Participants indicated that creative work is impossible for them when exploring a new framework or programming language. A firm grasp of the basics is required before being able to do any kind of creative work.

Some participants suggested that creativity says something about the person, not just about the work they do. They wondered whether a person with a big creative drive would be creative in any field, given a presence of necessary basic knowledge. Personality correlations with creativity have indeed been confirmed before (Reuter, Panksepp, Schnabel, Kellerhoff, Kempel, and Hennig, 2005).

Curiosity is cited as an indicator of creativity. A participant suggested that creativity is constantly looking to improve things. Whether or not a person is a continuous improver may be inferred from the frequency of new books on the desk. A "hungry" mind has been proven before to be a core determinant for achievement (Von Stumm, Hell, and Chamorro-Premuzic, 2011).

Context: physical, cognitive, and emotional

A safe environment, in which developers have the room to express themselves and where failing often is not punished. The more people are penalized for trying out something different, the less new propositions will emerge from them in the future. A lot of participants felt that they are more creative when they deliberately step *outside of their comfort zone*. This could mean presenting something for a non-technical audience, sharing knowledge, or trying to map problems and solutions outside of their own problem domain to their own. This requires a sufficiently safe environment.

A flexible environment that enables working whenever and wherever they want. This reduces stress and leaves more room for the actual (creative) work—as long as no other mind-numbing activities such as time tracking are required.

An interactive environment. Social interaction, supplemented by drawing material, increases the flow of information, which in turn makes that creative brew even more powerful. Sparring with peers of the same level further eases the exchange of info. Everyone confirmed that higher levels of creativity are reached together than alone—even if interaction is not directly created to creative processes: smalltalk and solutions offered as a joke to entertain each other may just as well lead to more creative work.

An environment that facilitates focus. This includes noise-canceling headphones and places where you can isolate yourself. Some stated they are not bothered by background noise or music while others are easily distracted by it, again hinting at the subjectivity of the circumstances to work creatively. Landscape offices are reported to be both a blessing and a curse: the flow of information is high, as are the amount of unwanted interrupts. Meinel, Maier, Wagner, and Voigt (2017) found that designing creativity-enhancing workspaces is by no means a simple task.

Leveraging productivity tools allows you to focus on the work itself, letting the tool do everything else such as boilerplating. Tools should match your own thinking process. In one focus group, we accidentally started a Vi versus Emacs war - that ended abruptly with one participant saying "*Vi won*".

Motivation to be creative

What motivates developers to be creative? We found several drivers we grouped in three themes: getting rewarded for their work, being cognitively challenged, and a personal need to be creative.

Reward

Although not everyone felt this was absolutely necessary, the customer-oriented aspect also plays a certain role. A solution to a problem should be *relevant* for end users, and be received

positively. As one participant expressed:

> *Doing something without feedback is just too non-committal, like working without context.*

Customer focus yields more satisfaction, since clients are the ones who provide the constraints on which you can unleash your creativity. Participants also think it is important to be given the responsibility to do major refactoring work. Appreciation for the work from colleagues and from clients is also highly regarded.

Challenge

Some developers actively seek out the borders of stepping outside their comfort zone, while others are forced to get creative by having to answer difficult "energizer questions": in both cases they are challenged which leads to creative work.

We noticed a similar importance of being challenged when one participant talked about his creative experience while working on a "brownfield" legacy conversion project, as opposed to a "greenfield" project where everything is written from scratch. Everyone agrees that greenfield projects enable creative thinking, since there is room to do things the way you want. However, brownfield projects introduce a lot of *constraints*, such as a strict budget and a legacy database that needs to be kept online, which in turn triggers more creative work (even though some developers dislike brownfield projects because they are more motivated to work on something brand new). Working with constraints has been mentioned several times during all sessions. Highly constrained conditions generate more ideas and are generally perceived as more inspiring (Biskjaer, Christensen, Friis-Olivarius, Abildgaard, Lundqvist, and Halskov, 2020).

Similar challenging aspects can be found in *Full-stack* projects. They are frequently mentioned as the most creative, involving the developer *in all of aspects of the design*: from back-end database structure to front-end UX design. Seeing the project work as a whole, with all pieces of the puzzle falling into place, seeking out input, and getting out of your comfort zone. These challenging projects usually come equipped with lots of constraints, for

example a migration or a hackathon with a limited time frame.

Personal need

Because otherwise it gets boring. Many developers love their job precisely because it requires a lot of daily creative work. Assembly line work was frequently mentioned as being horribly dull. Nonetheless, someone suggested that a few days of repetitive work might even stimulate creativity. After mastering something, they start wondering how to automate things just to give in to their creative urge:

> *In a sense, creativity can work therapeutically. I wouldn't be here if this work wasn't creative.*

Tools and techniques

To enable and improve creative work, various tools and techniques are brought forward. They can be categorized under analogies and (external/internal) feedback.

Analogies

Mapping solutions from another domain to your problem field is one of the more intriguing techniques. For instance, if you are developing a virtual communication channel, you could try to investigate how these problems are already solved in the postal system or face-to-face conversations.

Feedback

As one participant creatively stated:

> *Creativity is the brew of different inputs.*

He actively tried to *seek out these inputs* as much as possible. Here, external feedback can be obtained directly by asking for it, but also indirectly by developing your own knowledge relevant for your work. There are myriad ways to do this, for example:

- Asking for a second opinion from others, not waiting until the code review.

- Regularly reading relevant literature to stay up-to-date.
- Attending knowledge sharing meetings.
- Sparring with peers, getting further together.

Next to external feedback, it is also possible to develop internal, self-reflective feedback, such as:

- *Peeling the onion* by keeping on asking "why".
- Arguing with yourself (*rubber ducking*).
- *Distancing yourself* from your thoughts, thereby approaching the problem from other angles.
- Seeking out edge cases like inventing improbable scenarios to undermine your own train of thought.
- Switching gears by zooming out to get the broad picture or zooming in on one specific aspect of the problem.

And of course, brainstorming and modeling the domain on a whiteboard was mentioned countless times.

Taking conscious breaks also came up often. Some developers were familiar with Hunt's (2008) "Pragmatic Thinking & Learning", in which he characterizes the well-known "shower thoughts" as asynchronous callbacks from *R*-mode thinking. Taking a coffee break or deliberately going to the toilet also seemed to pay off.

Creativity as a value for SE

Although creativity is considered an important skill in SE, we found that it is more valued when combined with aspects of work commitment and critical thinking.

Combined with commitment

In one group, there was a heated discussion about bug-fixing. Some claim finding the bug is creative work, but not fixing it, while others say it is the other way around. It also seemed to depend on the type of bug, and who caused it. Of course, it takes some time to understand someone else's thought process. Some developers do not like to dig deep if they did not write that piece of code, which is

why they dismiss debugging as not very creative, while others who like to get their hands dirty seem to claim the opposite. Similar discussions on starting from scratch or not can be found in 'greenfield' versus 'brownfield' projects: some developers dislike brownfield projects because they are more motivated to work on something brand new. It is clear that the ability to be creative combined with a commitment to "dig deeper" allows for more successful work outcomes which employers supposedly value more.

Everyone agreed that being creative is a requirement to successfully tackle complex problems. Experts noted that most domain-specific problems are not solved before, as the context and limitations of problems are almost always unique. Some participants felt the need to distinguish themselves by committing to be more creative than others. Others say that creativity happens intuitively:

Creativity simply arises when you are solving a problem.

Many developers are passionate in what they do. They can unleash their creative drive at work by showing craftsmanship. However, participants also mentioned not everyone has this urge, and that is fine too. Developers can also solve certain problems by simply using their experience of previously encountered problems, instead of always trying to be creative. There is a time and place for creativity.

Combined with critical thinking

It is important to note that creativity does not always have a positive connotation. One can come up with extremely creative, but completely unusable solutions. Participants emphasize the right combination between creativity and critical thinking, taking into account the context and constraints of the problem:

Creativity is the means, not the goal.

Measuring creativity

To jump-start the discussion how to measure creativity, we provided the example of the interview process when applying for a job in the SE industry. How can you measure whether an applicant is creative? According to participants, by *gauging the thought process* when a problem is presented. We list a selection of the possibilities:

- How are problems approached? Present an impossible scenario and see how far they get in trying to solve it.
- Ask questions about something unfamiliar to the applicant.
- Ask open-ended questions outside of the SE field.
- Do a tunnel vision test: ask to list (unit) test cases besides the usual suspects. Are all edge cases considered?
- Is the applicant a critical thinker, and if so, does he only utter critique or also come up with alternatives?
- The wheel does not always have to be reinvented. Does the applicant know the *DRY* and *YAGNI* principles?

Participants additionally mentioned puzzling games such as Escape Rooms and Black Stories as great ways to test creative thinking. Another option is taking a visual approach by looking at the portfolio of the applicant. However, the general consensus was that it is difficult to measure: all indicators are highly subjective.

The same is true for evaluating whether colleagues are creative. Someone suggested to look at body language. Are they happy, and making a lot of jokes? Are they "in the zone"? A participant rejected that statement, as he claimed one can also be very much in the zone by simply "sticking stamps". After some discussion, the conclusion was as follows: do they pause now and then, perhaps thinking? If the pause is too long, they are stuck. If there is no pause, it is likely to be assembly work and not creative work. Measuring productivity (visible, doing) is something else than measuring creativity (invisible, thinking). As one developer stated:

> *Most of my creative work happens in the car. When I'm at work, all I have to do is type out the solution in my head.*

Next to the earlier described curious and "hungry" mind, encouraging others to think creatively by frequently engaging in

the discussion is also seen as an indicator for creativity. According to participants, communication is an important aspect of creative work in SE. Everyone agreed on the following:

An open-ended question should trigger something in a creative person.

To conclude, observing certain behavior could be a precondition for creativity, not a guarantee. It is easier to evaluate whether a product is creative than whether the process is creative[2]. When asked if effectiveness is linked to creativity, someone commented that it is not a requirement, however, in SE, effectiveness is usually an important constraint.

[2] Chapter 11 proposes a new assessment technique to measure concepts involved in the creative progress. We will leave the discussion of existing measurement techniques until then.

9.3 Limitations

THE QUALITATIVE NATURE OF this study makes it difficult to provide a unifying definition of creativity applicable to everyone in SE. However, defining creativity was never the goal of this research. By adhering to the methodology and by cross-validating data, as explained in Section 9.1, we believe that the results provide relevant and interesting insights that enlightens the role of creativity.

Also, since creativity is a context sensitive and very personal matter, generalizations prove to be very challenging. This could be mitigated by further reproducing this study in other countries and companies. We are convinced to have collected enough data to contribute to creativity research.

As mentioned by some participants, only creative people are attracted to join the focus groups. We intentionally selected on this intrinsic interest to increase relevance of answers. In this way, we could answer `RQ1` and `RQ2`. However, it is still unclear how much creativity is applied daily in the field of SE. Not every discussed form of creativity (`RQ1`) is seen as desirable or as making one developer better than the other (`RQ2`). For example, the creative one who continuously looks for new ways without effectively delivering a final product may be a worse developer.

9.4 Conclusion

The 33 participants of four focus groups brainstormed about the role of creativity in SE, resulting in seven distinct themes, as displayed in Figure 9.1: *technical knowledge*, *communication*, *constraints*, *critical thinking*, *curiosity*, *creative state of mind*, and *creative techniques*. These themes and subthemes also provide insight into how creativity could be measured, the reasons for being creative, and what environment is needed to enhance creative work.

There seem to be different levels of creative drive: some software engineers are more passionate than others, both in their work and beyond. Regarding the domain-specificity of creativity (Silvia, Kaufman, and Pretz, 2009), we feel that most techniques could also be applied to architecture for example. Of course, both software design and architecture design share the word *design*. The creative urge thrives in the unexplored: "*it is the unknown that is the most creative*".

We hope this chapter is useful for two particular groups. First, for SE practitioners, who are looking for practical tips on creative problem solving. As discussed in Section 9.2, each of the seven identified creative dimensions can prove to be an effective tool when facing a problem in the SE world. Second, for the computing education community, who aspire to inject more creativity into their curriculum.

Since our ultimate goal is to imbue and further encourage the creative urge among undergraduate students in higher education, repeating this focus group study with students instead of professionals could yield interesting results that complement the findings presented above. Thus, in the next chapter, we explore the perceived creativity of SE students and present a comparative study to further uncover the creativity skill gap in SE education.

10 Students' Perceptions of Creativity

Publication details

This chapter has been accepted in the Taylor & Francis *European Journal of Engineering Education* as *Students' and Professionals' Perceived Creativity In Software Engineering: A Comparative Study* (Groeneveld et al., in press) and will be published shortly.
Its main contribution is the discovery of fundamental differences of perceived creativity between students and professionals, grouped into five themes: *the creative environment*, *application of techniques*, *creative collaboration*, *nature vs nurture*, and *the perceived value of creativity*. The paper also summarizes implications for educational practice.

The seven dimensions of creativity uncovered in Chapter 9 depict a clear picture of how professional software engineers perceive creativity in their day-to-day activities. Compared to existing literature on creativity in higher education, we suspect that students' perceptions do not neatly align with that of professionals.

Therefore, this chapter digs deeper into the awareness of creativity from the viewpoint of the SE student. We have conducted 10 in-depth interviews using the same analysis as presented in Chapter 9 to help answer the following questions:

Chapter Research Questions

1. *How do professionals perceive creativity in the field of SE?*
2. *How do students preceive creativity in the field of SE?*
3. *What are the similarities and differences in perception of creativity between both worlds?*

RQ1 was already answered in the previous chapter but will be revisited for RQ3, in order to compare the results of RQ1 and RQ2. Before delving into the comparative methodology and results, we first provide a brief overview of existing work that explored perceived differences in creativity.

10.1 *Perceiving (differences in) creativity*

The students' understanding of mathematical creativity was explored by Catarino, Nascimento, Morais, Campos, and Vasco (2019), by sending out a survey to 128 engineering undergraduate students and asking them "What do you understand by mathematical creativity?". The result categories were *involving mathematics* (the application of mathematics in other areas; connections between mathematical concepts), *problem solving* (multiple or original ways to solve a problem), and *out of the box thinking*. Perhaps this hints at certain domain-general aspects of creativity, versus domain-specific aspects related to only mathematics or SE. The authors conclude with a remark that, compared to other related works, the perception of their surveyed first-year students might differ substantially from that of graduate students.

Bjørner and Kofoed (2013) approached the perception of creativity differently, by interviewing academic teachers instead of students. The researchers were interested in how cross-disciplinary teachers perceive and facilitate creativity. Again, the results point towards problem-based learning and problem solving as an important element for the creative process. The interviewed teachers had very different definitions of creativity, but they all

emphasized the social context (in this case, the cultural learning environment) as an important factor, both at macro and micro level. Teachers also frequently link motivation to creativity. Bjørner and Kofoed end with a call for action: teachers seem to lack proper support and tools to facilitate creativity in education.

Mohanani, Ram, Lasisi, Ralph, and Turhan (2017) investigated perceptions of creativity in SE research and practitioners by comparing a systematic mapping of research literature with 17 interviews of professionals. They discovered some agreement—Kaufman and Sternberg's (2007) novelty and usefulness—but more importantly identified differences in the way creativity is conceptualized, measured, and improved. Their research sheds some light on the research/practice gap but does not directly include the perception of students.

To our best knowledge, no study asks both programming professionals and students what they think creativity is like in context of problem solving. Our contribution complements the above works, as will be further discussed in Section 10.3.

10.2 Methodology

This section describes the approach used to answer RQ1 to RQ3. The study was executed in three phases: gathering data from professionals using focus groups, gathering data from students using semi-structured interviews, and comparing these findings.

The approach for the first phase can be found in Section 9.1. The transcripts analyzed in Chapter 9 were re-analyzed in context of the current chapter. Figure 10.1 puts the previous and this current work into context.

Interviewing students

The focus group study was a first step towards exploring the role of creativity in SE. During academic year 2021–2022, we started a campaign to enlist students with the intention to replicate the 2021 study using exactly the same methodology but with different subjects: university students. However, limited engagement of the

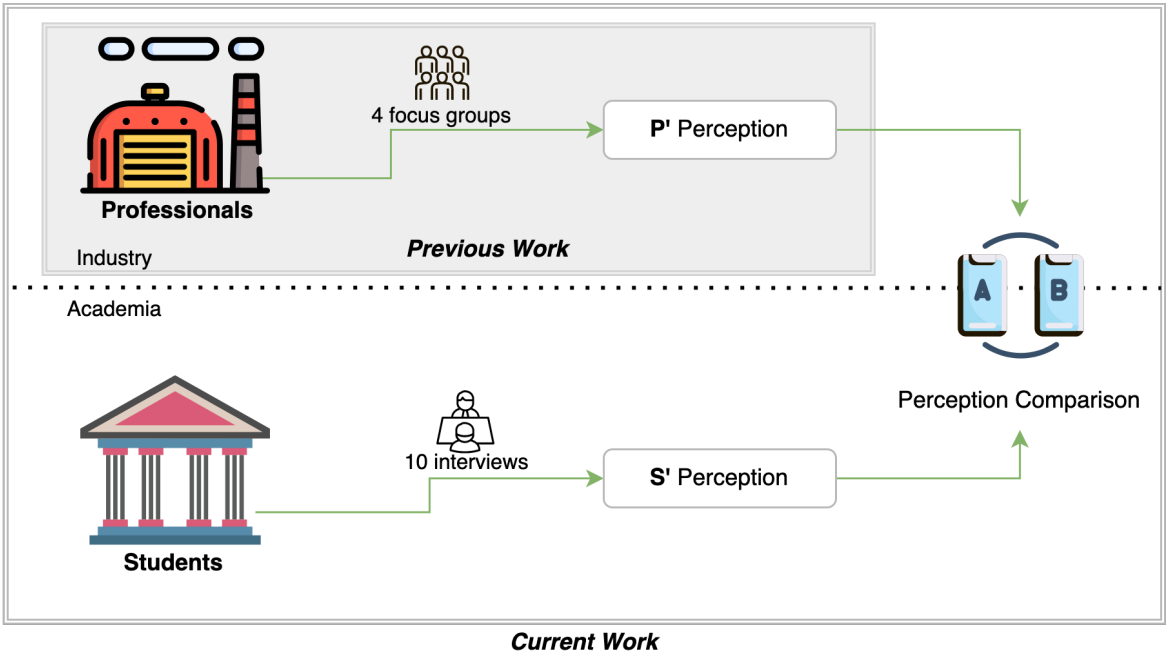


Figure 10.1: The context of previous and current works.

students and planning issues to get everyone together forced us to slightly revise the working method.

Instead of organizing focus groups, we interviewed students one-on-one. The questions were the same as the ones answered by the professionals but slightly altered to better fit the context. For example, "How can you see when a *colleague* is being creative" became "How can you see when a *fellow student* is being creative"[1]. To keep the conversation going, we asked follow-up and clarification questions about responses we found intriguing (e.g., novel, vague, avoiding or in contradiction to prior answers), again in accordance with Hove and Anda. Although interviewees this time did not have the chance to brainstorm in group before answering, we still gave them ample time to respond.

In total, 10 students were interviewed across 4 different universities in the same North-European country, of which 5 undergraduate and 5 graduate students. To increase participation

[1] The questions posed remained the same as in the focus group study and can be found in Section 9.1.

rates, students could secure a video game voucher of `60 EUR` with a `1/7` chance of winning. Neighboring universities that provide a computer science course or an engineering course with specialization in SE were asked to spread the announcement in the form of a landing webpage via their channels. All interviews were conducted in the students' native language to keep the conversation fluid.

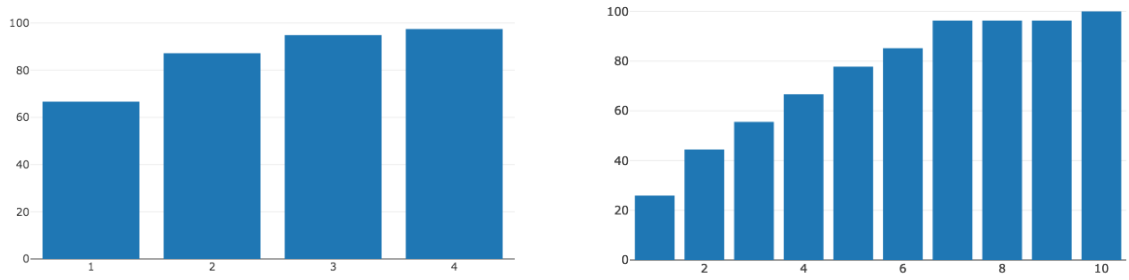
Analyzing recorded data

Our intention was to treat, and thus analyze, the data in the same way, for both groups of subjects (professionals and students). The data was once again processed using the qualitative coding techniques described by Onwuegbuzie et al. (2009) and Richards and Hemphill (2018). After audio recordings were transcribed, the transcript and written notes were read multiple times to apply an open coding step. This was done by the first two authors independently in order to let patterns "bubble up" in an emergent-systematic fashion as suggested by Onwuegbuzie et al. Next, notes were compared and cross-validated, followed by an axial coding categorization step. After making multiple revisions of mind maps and merging the results of all focus groups and interviews, we finally arrived at 45 codes divided into 5 categories.

Thematic data saturation was calculated by counting the number of discovered codes for each interview or focus group. As Figure 10.2 indicates, after the fourth focus group and the tenth interview, less than 5% new codes were introduced, which is a good pre-set threshold according to Guest et al. (2020).

Comparing findings

Grounded theory dictates a bottom-up approach to identifying common themes in qualitative data. While the application of this approach for both datasets, as seen in Table 10.2, proved to be a good exercise, it ultimately failed to zoom in on the differences between both groups. We felt that the axial coding step used to reduce the number of subthemes is partially responsible for the loss of relevant subtleties and context. For example, both



(a) Focus groups with professionals.

(b) Interviews with students.

Figure 10.2: Indications of thematic data saturation: interviews (x-axis) projected on the percentage of identified codes (y-axis).

professionals and students mentioned working with (heterogeneous) groups, which can be seen as a way to operationalize creativity—the "how"—but also as a physical setting—the "where". Furthermore, while it was mentioned by students, the way in which they mentioned it made clear that they were mostly theorizing ("I guess?"), compared to professionals who really lean on collaborative efforts. Another identified subtheme, actively seeking out input, appeared as a central theme in focus groups, but merely as an aside in the student interviews.

To better highlight these subtle but very relevant systemic differences, we reverted from a bottom-up approach to a more top-down approach, zooming in on the differences that appeared in the first pass while identifying codes bottom-up. While re-reading the manuscripts and re-executing the coding steps a third time, we paid special attention to the following concepts:

- The difference in environment in which the creativity is executed (professional vs a learning environment);
- The awareness and application of creative techniques: by chance or on purpose;
- Collaborative aspects of creativity and its perception;
- The perceived role of nature vs nurture in the ability to be creative;
- The perceived value of creativity in the profession or the project at hand.

In Section 10.3, we will provide a brief overview of the bottom-up findings, before zooming into the comparative

similarities or differences, described in Section 10.4.

10.3 Bottom-up results

TABLE 10.1 CONTAINS SUMMARIES OF gathered interview metadata, similar to Table 9.1 for focus group metadata in the previous chapter. When totaling both, 43 participants generated 89.462 words in 652 minutes worth of transcripts. On average, a focus group session with professionals—of on average 8 participants—lasted 100 minutes, while an interview session with students took 25 minutes.

The full data set of anonymized transcripts and source code used to process these are available at https://doi.org/10.48804/3FHVRC.

Description	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Duration	23m	25m	22m	22m	24m	31m	24m	24m	23m	35m
Words	3 321	4 211	2 722	3 028	3 533	4 802	3 812	2 713	4 247	4 690
Theme distr.	26%	44%	56%	67%	78%	85%	96%	96%	96%	100%

Table 10.1: One-on-one interview metadata of 10 students, totaling 37.133 words and 253 minutes of data. Theme distribution rates are also visible in Figure 10.2.

The following subsections provide a summary of the emerged subthemes and categories while bottom-up analyzing the transcripts of professionals and students. Themes are marked in bold. We will zoom in on the top-down approach in Section 10.4.

Table 10.2 contains the identified codes that were grouped during the first analysis pass as described in Section 10.2, including a presence marker of the code in either the transcripts of the professionals and/or the students. Some codes are a good fit in multiple categories (e.g. linking with other domains could belong in either *when* or *what*).

When are you (not) creative?

Perhaps unsurprisingly, most professionals and students agree that creativity is needed to come up with a **unique solution** to a new problem. This problem should be **challenging**, yet at the same time not **too difficult**. If it is too easy, it is considered as mundane

Subthemes occurring in groups:		P.	S.
Not yet a solution, a unique/new problem	**When?** (9)	✓	✓
Slept well, in the mood			✓
A challenge that dictates thinking		✓	✓
Not too difficult			✓
Linking with other domains		✓	✓
Creating something visual		✓	✓
Freedom of choice			✓
Contribute something to society			✓
Performance matters		✓	✓
Quality matters	**Why?** (9)	✓	✓
Overcoming constraints		✓	✓
Client-oriented thinking		✓	
Innovation/renovation is needed		✓	✓
Because otherwise things get boring		✓	✓
Personal satisfaction		✓	✓
That's just the way I am		✓	✓
Fast feedback		✓	
Admiration for other creative work		✓	
Subjective; taste-dependent	**Why?** (7)	✓	✓
Studying their reasoning		✓	✓
Counting the amount of different solutions		✓	✓
Judge the end result		✓	✓
Check for elegance in code		✓	✓
Shared responsibility		✓	
Body language; surprise		✓	✓
Pseudocode rendering	**How?** (11)		✓
Divide and conquer			✓
Drawing diagrams, domain modeling		✓	✓
Continuous learning		✓	
Actively seeking out new input/inspiration		✓	✓
Discussions/brainstorming (on whiteboard)		✓	✓
Exploring different angles/bird's eye view		✓	
Ask yourself questions: rubber ducking		✓	
(Dare to) ask others questions		✓	✓
Dare to discover			✓
Get out of comfort zone; be open to new things		✓	✓
Tooling support	**Where?** (9)	✓	✓
Support from your environment		✓	✓
Being in a flow state		✓	✓
Individually behind the PC		✓	
In a (heterogeneous) group		✓	✓
In a quiet environment		✓	✓
Music/solitude		✓	✓
Sport/walking to take mind off of things		✓	✓
Deliberate breaks to trigger "aha!" moment		✓	

Table 10.2: Emerged subthemes of professionals (`P.`) and students (`S.`).

and boring, and creativity rapidly wanes. If the problem is perceived as too difficult—for instance if the API or programming language is completely new—participants first try to learn and follow existing rules before attempting to tackle it creatively. According to Csikszentmihalyi (1997), just the right amount of difficulty is important to achieving a creative flow-like state.

Students put heavy emphasis on **freedom of choice**: they report being the most creative if they have full control over the assignment such as choice of framework and implementation method. On the other hand, when a course project imposes a certain technology and is less open, they consider it to provoke less creativity. This goes against the grain of professionals' coding environments that are usually filled to the brim with imposed constraints. We will explore this more in detail in a later section.

For some participants, working on **something visual** like the UI design or making drawings on a whiteboard is perceived as being more creative. For others, their creativity sparks when **performance** is a factor, again hinting at the importance of both imposed and self-imposed constraints in creative problem solving.

One student jokingly mentioned that **sleeping well** and being in the right mood also determines when they are more or less creative. This lines up with the literature that suggests that sleep does indeed facilitate creativity (Ritter, Strick, Bos, Van Baaren, and Dijksterhuis, 2012).

During the warm-up questions, some professionals attempted to define creativity:

> When someone can come up with an elegant solution to a previously unsolved non-trivial problem.

This turned out to be quite challenging, as the definition was almost immediately heavily disputed. All students and professionals did agree on the fact that creativity is inherently personal. Some students suggested it was subjective and depending on the person evaluating the creativity, which lines up neatly with the attention that is currently paid in cognitive psychology to the socio-cultural context of creativity.

Why should you be creative?

When asked why creativity matters, interviewees unanimously answered because **quality matters** and can only be achieved by staying longer with the problem, coming up with alternatives, poking at the system from different viewpoints, again **overcoming constraints** that obstruct the path towards creation.

Besides conventional answers, such as "because **innovation requires creativity**", both students and professionals hint at the **personal satisfaction** of approaching coding problems creatively. Without creativity, things become boring fast, according to respondents:

> In a sense, creativity can work therapeutically. I wouldn't be here if this work wasn't creative.

Recent work by Graziotin, Wang, and Abrahamsson (2014) revealed that happy programmers perform better, and thus might be more creative. Some participants went as far as saying that creativity **is part of their personality**, of who they are: a *Homo Faber*, Umberto Eco's concept of the urge to use tools and manipulate technology to create and shape the world around us (Eco et al., 1989).

Assembly line work—or in programmers terms, generating getters and setters and doing the plumbing setup work of a project—was frequently mentioned as being horribly dull. Still, some professionals like to pick up mundane tasks to trigger later creative "aha" moments, while students rejected this kind of work all-together.

The "why" theme again hints at dissimilarities between students' and professionals' opinions: professionals are tuned to more iterative software development processes that facilitate **fast feedback** and **client-oriented thinking**. Within the safe walls of the university, this is completely absent and can only be partially emulated. Professionals also more openly **admire others' creative work** which in turn sparks their creative skills. One student did mention that seeing other students at work on a similar exercise can help sparking creativity. However, for professionals, finding a creative muse goes beyond similar work: some even marveled at

the engineering work of suspension bridges or highway interchanges.

What is creativity like?

Besides the **subjective** nature of creativity that was brought up multiple times, when we asked interviewees how creativity could be recognized and measured, many thought it could be observed by **following the reasoning** of others after asking them an open-ended question. If they deliberately think of alternatives, critically evaluate options, sometimes backtrack, sometimes zoom in or zoom out, then their (software development) thought process is deemed as creative.

Both parties also identified the conventional answer: assessing **divergent thinking** by counting the number of solutions that get brought up. The more solutions, the more creative. In essence, this is Torrance's (1972) TTCT[2], that contemporary creativity research is moving away from (Glaveanu et al., 2020), as divergent thinking is only one aspect of creative problem solving.

[2] See Figure 7.1 of Chapter 7 for an illustration of a typical TTCT assessment.

Instead of trying to judge the creativity level of the development process, interviewees suggested to **evaluate the end result**: the product itself. The evaluation of a product as judged by an expert group is called Amabile's (1982) CAT, and according to the literature reviews mentioned in Chapter 8, it is an often used technique in SE education.

Both professionals and students mentioned **elegance in code**, but both interpret this differently: students focus much more on complexity, perhaps to show off their skills as part of the assignment, while professionals put emphasis on creative clean code (and the many unit and integration tests that come with it) to make long-term maintenance much more bearable.

Only professionals mentioned **sharing responsibility** as an integral part of creative problem solving in SE. Of course, the culture and work environment differs radically compared to SE students in higher education. A detailed comparison of different social and cultural creative contexts is described in Section 10.4.

How can creativity be induced?

The most striking difference between both study groups is the number of creative techniques that are brought up during interviews and focus groups. Experienced programmers clearly are at an advantage here, mentioning techniques such as **shifting angles** (going back and forth between zoomed-out software architecture and zoomed-in code at method level) and **rubber ducking** (literally arguing with a rubber duck—yourself—when stuck to force a shift in perspective). Some techniques such as quickly drafting a potential solution in **pseudocode** and **dividing** a problem into small sub-problems were mentioned by students but not by professionals. We argue that they are so commonplace in industry that they are hardly worth mentioning at all, while for students, they are still a fairly novel practice.

Both parties mentioned getting **out of your comfort zone** and **daring to ask questions**: if you are not open to learning new things, your creativity will be barely fueled to keep on going. Feedback and collecting information is important, both from people (mentioned more often by professionals) and in the form of read-only information found in books, websites, or other sources (mentioned more often by students). One professional summarized it neatly as follows:

> *Creativity is the brew of different inputs.*

The concept of **continuous learning** is well-known within the world of SE (Rodrigues da Fonseca, Silva, Silva, and Pereira, 2019), yet still fairly new to students. They are still steeped in a very supportive learning environment, while in industry, it takes deliberate practice to keep on being involved in learning, making it a more important factor for creative success.

Where are you the most creative?

All interviewees emphasize the need for a **supportive environment** in which mistakes are not immediately punished. For students, this means dealing with open assignments that do not have a one size fits all answer, while for professionals, this means management that allows room for creative experiments.

Some programmers seemed to prefer working with **music** to have easy access to Csikszentmihalyi's **flow state of mind** while others prefer a **quiet** environment. SE students interpret that quiet environment as literally **being alone** in front of a PC and although both study groups mentioned **working in group**, this was much more ingrained in the professionals' answers.

Generating creative ideas does not need to happen exclusively clutched in front of a screen. **Taking a walk** or engaging in other activities like sports to let the mind wander and subconsciously think about a complex problem was mentioned by many respondents of both groups. Existing research such as that of Colzato, Szapora Ozturk, Pannekoek, and Hommel (2013) supports the notion that physical movement enhances divergent thinking. The "aha moment" usually does not occur when writing code but while doing something else. In fact, most of the thinking work can be done somewhere else, as one professional stated:

> *Most of my creative work happens in the car. When I'm at work, all I have to do is type out the solution in my head.*

Interestingly, professionals seemed to deliberately trigger these moments while students recognize them when they occur but fail to understand and apply it as a creative technique.

In the next session, we will discuss these differences in detail with the help of the results of our top-down approach.

10.4 Top-down results

The previous section presented the results of a thorough bottom-up analysis of the transcripts that revealed the when, why, what, how, and where of creativity as an important factor in problem solving for software engineers. As indicated by the interview occurrence columns in Table 10.2, professionals and students did not always agree. This section zooms in on those details and highlights both subtle differences in interpretation and bigger gaps in the perception of creativity. We end the discussion with a set of implications for educational practice.

The mind map in Figure 10.3 summarizes the five central themes used in this top-down approach. It is important to note that these themes are highly interconnected, as emphasized by the arrows in the mind map. For instance, the way you interpret creativity alters your view on collaboration, or the environment that determines the company/university culture influences the application of certain creative techniques. We agree with Nerubasska and Maksymchuk (2020): creativity is *systemic*.

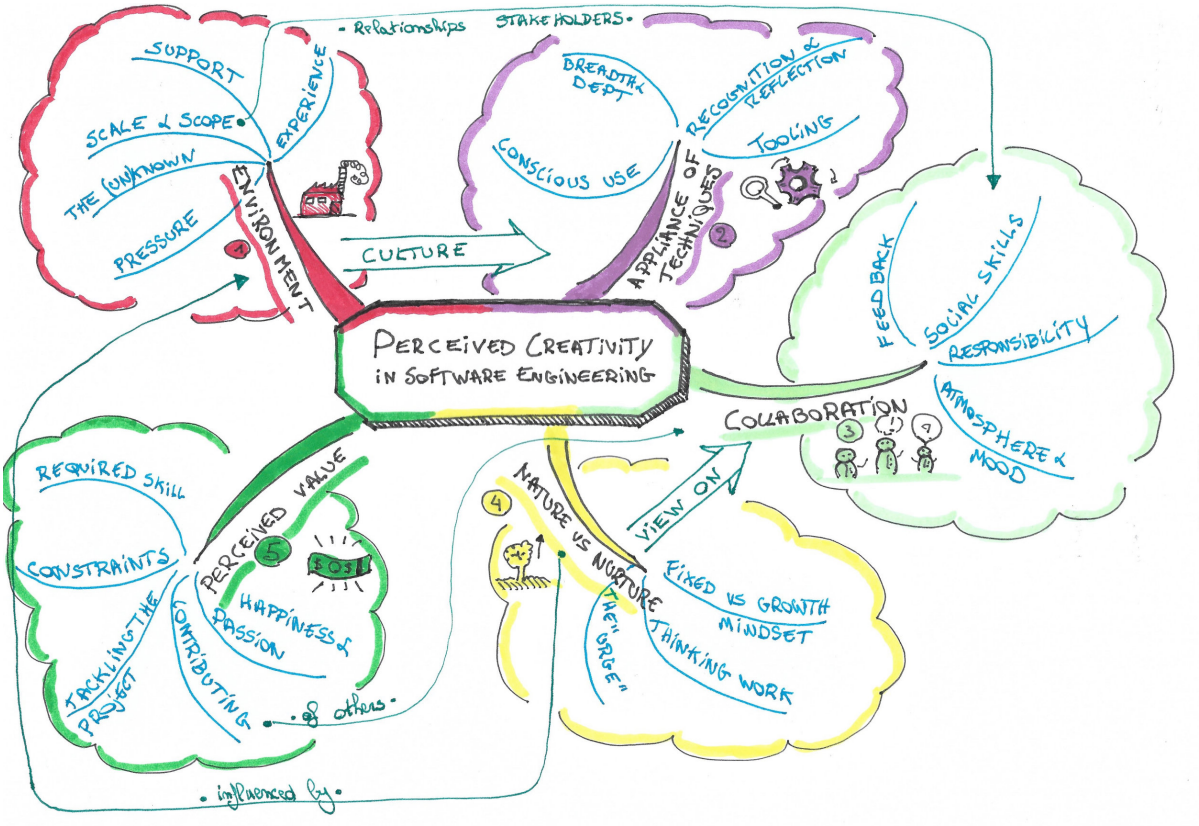


Figure 10.3: A mind map of the different perceptions of creativity in SE between both study groups.

Environmental differences

Professionals and students have to exert their creativity in very different environments, of which especially the **scale and scope** of both the projects and everything that surrounds these is vastly different. For instance, the much larger development environment in industry includes:

- More context-switching between debugging, restructuring, and writing new code;
- DevOps activities such as continuous integration and distributed deployment;
- More emphasis on (automated) unit and end-to-end testing;
- Long-term legacy software support and possibly data migration steps to undertake;
- Much more frameworks and APIs to juggle;
- More data, logs, and requests to process;
- The involvement of many stakeholders that requires more interaction;
- ...

To a professional software engineer, creativity isn't just limited to writing code or fixing problems in code: it happens both at micro-level in code and at macro-level in the software architecture. As typical university assignments are very limited, even if they are open ended to help spark creativity (VanDeGrift, 2015), this part of creative problem solving is virtually unknown to SE students. Furthermore, the speed at which some frameworks come and go also requires a certain critical and creative judgement during mid to long term software development planning. Some professionals even mentioned dealing with patents. Experts testified that the more **experienced** you are working in these environments, the easier it becomes to express yourself creatively.

Another environmental difference is working with the **(un)known**. Many times, clients do not really know what they want, requiring a critical mindset, asking many questions, and short feedback cycles. We found that many students did not dare to question the question, which, as discussed in Chapter 9, is a component to creativity. In a sense, problem solving in SE is

related to "wicked" problems in design thinking, which Buchanan (1992) defines as "*as a class of social system problems which are ill-formulated, where the information is confusing, where there are many clients and decision-makers with conflicting values, and where the ramifications in the whole system are thoroughly confusing*". Wicked problems require creativity to successfully tackle.

Typical student projects exhibit some form of **pressure**, but this is usually limited to time pressure. Enterprise software development projects come with many levels of pressure: budgeting, time, expectations from peers and management, having to support existing legacy systems, ... Some of these can be classified as constraints which we will explore in one of the following subsections.

Creativity can only happen if the environment **supports it**: are peers and managers supportive? Does the company culture allow for a bit of play that might spark new ideas? Students interpreted support as coming from their friends and family, while professionals mention the bigger picture. For students, most projects have only one purpose: passing the course. Some admit to trying to optimize their grades. The fragmentation of typical SE curricula and the lack of feedback during and after the project does not work in favor of creativity.

The awareness and application of creative techniques

Creative techniques have to be **used consciously**. In the professional focus groups, one of the central themes was often consciously choosing when *not* to be creative: when writing a script that only gets executed once or when the solution can be found by repeating a previous practice and not time and time again questioning the architecture or trying to reinvent the wheel. None of the interviewed students mentioned this. A professional summarized this cleverly:

> *You [only] have to be creative when it is time to do so.*

Related to the conscious use of creativity is the **recognition of and reflection on** these creative actions. The context of the student interviews revealed that they might be capable of recognizing the

moments, but not yet of actively capitalizing on them. One student said:

> *I'll admit, I've never really looked for creativity. As is often the case, either [the solution] works, or it doesn't. Most of the time, I just sit here at my desk.*

Also, because of the much larger environmental scope, the creative techniques that professionals employ clearly showcase more **breadth and depth**. Students seem to be very much focused on the technical complexity of algorithms and the code on a micro level, while professionals mention a much more diverse set of creative techniques they utilize daily: switching between refactoring and redesigning, zooming in and out, visualizing problems, adding in a so-called "deliberate hack", taking edge cases into account when writing tests, ...

One of those examples that stood out was **tooling**. For SE students, tooling meant again getting to work with technical complexity: being creative by designing their own tooling systems. Interestingly, professionals use tools (e.g. editors and IDEs) to get rid of the unnecessary clutter and be able to focus on the essence: the coding problem at hand.

The lack of breadth and depth causes students to stop pushing forward by switching to other creative techniques when one fails to work. One student did mention writing many unit test cases to split a problem in small sub-problems, but did not mention the other potential creative advantage of testing that was mentioned by almost every professional: poking at the problem from multiple angles to arrive at an airtight implementation. This again signifies a lack of practical experience, which is of course to be expected.

Perceived collaborative aspects of creativity

As underscored by professionals:

> *For creativity, communication is key.*

Various **social skills** to help deal with the communicative and collaborative aspects of creativity are of importance as well, especially in a professional environment where different

stakeholders have different expectations of both developers and the project in development. Knowing how to approach others is key to getting the most creative ideas out of that interaction.

Another professional testified:

> *The ability to incorporate **feedback** of others in your solution-oriented thinking is of course an important part of the creative process.*

The importance of feedback is echoed by students, however, their feedback is much more limited. For some assignments, depending on the course or the teaching staff, they do not receive any feedback at all, severely hampering reflection on the creative process. Students also seemed to prefer getting inspired by other work through books or online resources instead of directly via discussions with their peers, hinting at a more individual development process. Students do mention that working in a diverse group will likely trigger more creativity, but the way they bring it up seems to indicate this as being purely theoretical knowledge:

> *People with a different background that might have followed different courses that I didn't take might possibly think about things I didn't think of. So I guess that's where the creativity probably surfaces then. I think.*

An existing body of literature indicates that interdisciplinary SE projects are nothing new, although they almost never focus on improving the creativity by taking advantage of the heterogeneous student groups. In one study, interdisciplinary computing classes are said to improve students' soft skills, but creativity was not mentioned (Carter, 2014).

Professionals also think it is important to be given **shared responsibility** to execute the planned work. This creates a unique **atmosphere** that, according to our participants, amplifies creativity. This sense of responsibility is absent with students: their only responsibility is submitting the work on time to hopefully pass and eventually graduate. Especially when closely working together, atmosphere and collegiality does have an impact on mood, which has been proven before to greatly affect the willingness to be creative.

Creativity can be contagious. Professionals often strategically place newcomers next to creative enthusiasts to help them quickly get up to speed. Students acknowledge this, but again, it sounds like theory that was never put into practice:

> [...] *I didn't think of that yet, usually that should also lead to new insights for yourself, I guess, when you see someone else doing something creative.*

Nature vs nurture: the creative mindset

While re-reading the transcripts, one thing that stood out was the **Fixed vs Growth mindset** of students vs professionals. Cognitive psychologist Carol Dweck describes a Fixed mindset as one that is rigid: either people are capable of something, or they are not; while a Growth mindset acknowledges the possibility to nurture skills (Dweck, 2008).

When it comes to creativity in programming, we observed primarily a Fixed mindset when interviewing students and primarily a Growth mindset when interviewing professionals. For professional software developers, continuous learning is part of their job, as every single component in their environment rapidly changes. They regard creative problem solving as a skill that can be learned, just like debugging or database maintenance. On the other hand, students tend to categorize people into ones that are very creative and ones that are not:

> *There are people who are spontaneously creative. and who don't think about it, while I do, I do introspect when I want to make something, and think about my work, so I'm consciously working on that.*

That person continued to describe behavior of friends who he thought were more creative, and fellow students who in his mind weren't. While "consciously working on that" sounds like a Growth mindset, in his mind, some people are capable of it, while others are not. It did not occur to him that this can be trained. A few students mentioned creativity was related to intelligence and seemed to suggest that intelligence is also very much a fixed affair. Students' rigid mindset attitude towards creativity also surfaced in other focus group studies (Katz-Buonincontro, Hass, and Friedman, 2017).

A "sudden" creative idea is the result of a lot of **thinking work**. "*Being creative involves a lot of hard work*", testify several professionals. They continue by saying that hard work is best alternated with deliberate breaks or periods of mundane work to let the unconscious do its work, ultimately triggering that "aha" moment. Ideas never occur in a sudden daze of brilliance. As for students, they recognize that moment, for example during a long walk or another sports activity, but do not deliberately employ these techniques to let ideas bubble up more often. Some students even admitted to not "catching" the idea—for instance by writing it down—and promptly forgetting it. All students found repetition to be very much not creative, while professionals sometimes insert periods of repetitive work to explicitly trigger something else.

Many professionals described their creative endeavors as the result of their **creative urge**: the need to do something creative, Eco's *Homo Faber*. This was less distinctly present with students, although they also all acknowledge the need to be creative, but interpret that mostly as having creative freedom.

The perceived value of creativity

Both groups agree that creativity is a **required skill**: it is needed to find and solve non-trivial programming problems, to keep on digging while bug fixing, ... We summarized this as **tackling the project**.

An important aspect of problem solving is dealing with **constraints**. Compared to students, professionals face a plethora of constraints: time, budget, code, legacy, and so forth. For many professionals, self-imposed constraints are just as important as imposed constraints, as they usually trigger creative thinking. Self-imposed constraints are an often-used technique in art and interaction design (Biskjaer and Halskov, 2011). For many professionals, working on a *brownfield* project—where existing software architectures have to be taken into account—was considered more creative than a *greenfield project*—where a team can start from scratch and use the latest tech—although this was sometimes debated.

Students do mention constraints but in a much more limited

sense. For most students, constraints work against creativity. One student noted that they preferred dynamic programming languages over static ones "*where everything is set in stone*", failing to see that exactly those limited environments breed the most creative solutions. Every student preferred a greenfield project, working from scratch, to be in full creative control. Only one student—who turned out to be working part-time for two years in the SE industry—mentioned self-imposed constraints.

Expressing oneself creatively can improve well-being, as expressed by several professionals and described in Section 10.3. For many professionals, writing code is seen as a **passion** or a craftmanship: pride is taken in delivering something of high quality, both on the outside (for the end users) as in the inside (clean code for the software maintainers). Students also express their fondness of creativity—that is, when an assignment allows some degree of creative freedom. Of course, for them, there is less incentive to write clean code beyond getting a grade: most code written in university is throwaway code.

Some students enthusiastically talked about their hobby coding projects where they are in complete control and can **contribute** to society by solving a problem that others also encounter. That "complete control" again highlights students' preference for working alone to cling to that precious freedom. Some students even expressed their aversion for working together as compromises have to be made which might muddle their creative vision.

10.5 Implications for educational practice

We are convinced that some of the identified differences in the perception of creativity can lead to actionable advice such as explicitly introducing creative problem solving techniques in SE courses. To summarize the results and discussion of the previous sections, we present a set of implications for educational practice that can be used to introduce and improve creativity in the classroom.

1. Help students discover the benefits of (self-imposed)

constraints[3], for example by introducing brownfield projects.
2. Keep the scale and scope of a typical SE project in mind when designing assignments, for example by involving more (imaginative) stakeholders.
3. Encourage creative explorations in problem solving, even if they do not lead to practical solutions, for example by focusing on the creative process in addition to grading the end result.
4. Plan for frequent moments of feedback, before, during, and after the submission of the final assignment.
5. Help students use a diverse set of creative techniques, including when *not* to be creative, for example by presenting multiple unique SE problem cases.
6. Show students that creativity is not a static personal characteristic but a skill that can evolve.
7. Emphasize the power of collaborative idea generation and iterative creativity, for example by introducing and actively coaching more group work.

[3] A whole chapter of *The Creative Programmer* (Groeneveld, 2023) is devoted to the concept of constraints, where SE constraints are approached using practical examples. See the conclusion at page 263.

10.6 Limitations

This comparative study sought to answer the research questions using qualitative methods. As we have discussed, creativity is a subjective matter, and opinions of experts and students vastly differ, making it challenging to draw general conclusions. Repeating our study with other participants, for example from different companies or universities abroad, might produce different results. However, by carefully cross-validating data and adhering to the best-practices of qualitative research, as explained in the methodology section, we believe that we have mitigated the biggest risks. The data saturation rate indicates that the 4 focus groups and 10 interviews provided enough unique data to proceed with encoding and analyzing the results.

The interviews with students did not follow the exact same procedure as the focus groups. One-on-one interviewees might experience more pressure in trying to quickly answer questions compared to brainstorming in group, resulting in possibly

short-sighted answers. To minimize this effect, we compared encountered codes in student interviews with those from the focus groups and grouped them accordingly.

Another possible limitation to this study is selection bias. Professionals mentioned this as well: only the people that were intrinsically motivated to talk about creativity joined our focus groups. We intentionally selected on this intrinsic interest to increase the relevance of answers for answering `RQ1` and `RQ2`. Still, it remains uncertain how much creativity is applied on a day-to-day basis in the field of SE, or how many students deliberately make use of creative techniques when working through their assignments.

Lastly, one of our initial aims for this research was to discover potential differences in the perception of creativity between undergraduate and graduate students. While it was clear—and perhaps expected—that students with some work experience started to lean towards the opinion of professionals, there was not enough data to make any claims regarding an evolving view of creativity as students mature during their education. Future research might shed more light on this.

10.7 Conclusion

THIS CHAPTER SOUGHT TO explore SE students' perceptions of creativity, and to compare these to professionals' perceptions that were discussed in the previous chapter. For this, we recruited 10 students in higher education who major in SE. Five themes emerged from a bottom-up qualitative code analysis: *when to be creative* (9 subthemes), *why should you be creative* (9), *what is creativity like* (7), *how to be creative* (11), and *where does it manifest* (9).

Next to these themes, a follow-up top-down analysis zoomed in on the similarities and differences between both study groups. The comparison, yielding 21 new subthemes, was summarized using the following five themes: *the creative environment* (5 subthemes), *application of techniques* (4), *creative collaboration* (4), *nature vs*

nurture (3), and *the perceived value of creativity* (4).

This chapters contributes to the computing education research community by helping identify the gap between SE industry and academia, specifically in context of creativity and creative problem solving.

As a next step, one could experiment with various creative interventions and measure the effectiveness of creativity teaching techniques. But first, we have to tackle the question of how to measure SE students' creative problem solving progress. The next chapter reveals a newly developed assessment tool to do exactly that.

11 Measuring Creativity

Publication details

This chapter has been published in the *Proceedings of the 2022 ACM Conference on Innovation and Technology in Computer Science Education* as *Self-Assessing Creative Problem Solving for Aspiring Software Developers: A Pilot Study* (Groeneveld et al., 2022c).
Its main contribution is the development of a self-assessment tool for computing students in higher education to measure their creative programming skills, which encompasses the seven dimensions of creativity as described in Chapter 9. The tool, called Creative Programming Problem Solving Test (CPPST), was psychometrically validated with the help of a pilot study ($n = 269$).

In Chapter 4, we discovered that industry experts perceive *creativity* as an important non-technical skill for software engineers. Nevertheless, according to Chapter 5, current curricula pay relatively little explicit attention to this skill. This may be partly due to complex and broad nature of the concept creativity, which has not yet been widely researched in the field of SE, as discussed in Chapter 7.

In this chapter, we are interested in the concept of Creative Problem Solving (CPS), i.e. the use of creativity to solve day-to-day programming problems. We build on the work as presented in Chapter 9, that examined this concept by means of a Focus Group

Study (FGS), which we will further refer to simply as *the FGS*[1].

Our goal is to make the insights from the FGS actionable for computing education research[2]. In particular, we derive from the study presented in Chapter 9 a creativity self-assessment tool for SE students, called the *Creative Programming Problem Solving Test* (CPPST). It allows students to measure their skills in each of the seven creativity domains. This may be useful for students themselves, e.g., as a means of tracking their progress throughout a curriculum, but it is mainly intended as an instrument for educational researchers who wish to investigate the effect that certain changes to the curriculum or to a particular course have on the students' creative skills.

Before examining the survey design and presenting the results, we first briefly discuss existing similar works that also attempt to measure creativity.

11.1 Background

In general, creativity is often measured through divergent thinking, as is the case for the popular Torrance's (1972) TTCT. While the TTCT can be used to assess *creative potential* (Kim, 2020), it does not attempt to measure the *creative process* itself (Miller, 2014a). To address this, Miller (2014a) developed the Cognitive Processes Associated with Creativity (CPAC) scale, focusing on the processes of brainstorming, metaphorical and analogical thinking, perspective-taking, imagery, incubation, and flow. The FGS also identified these topics, but, in addition, the industry professionals also stressed topics such as constraint-based thinking and communication (see Section 11.2). In the SE context, the CPAC scale is therefore only part of the story.

The same can be said for similar efforts of measuring creativity in general, such as the Creativity Support Index (CSI) of Carroll, Latulipe, Fung, and Terry (2009). It consists of the components: exploration, expressiveness, immersion, effort/results, enjoyment, collaboration. This list also lacks several important topics from the FGS. However, the *expressiveness* component was not mentioned in

[1] As a reminder, the FGS identified 39 themes that are relevant to CPS, grouped into seven domains: *technical knowledge*, *communication*, *constraints*, *critical thinking*, *curiosity*, a *creative state of mind*, *creative techniques*. For more information, see Chapter 9.

[2] The reason why we focus on the FGS instead of the findings of Chapter 10 is that we are interested in bridging the gap between industry and academia by focusing on what industry experts think is relevant, and then amplifying these creative skills with students in education. Therefore, the CPPST should be based on professionals' opinions, not that of students.

the FGS: while self-expression is often thought to be a key part of creativity in general, industry experts do not appear to consider it as an important component of CPS in software engineering.

Scales have also been developed to measure creativity specifically in engineering students. Amato-Henderson, Kemppainen, and Hein (2011) based their scale on three existing scales: the Resistance to Change scale, the Curiosity and Exploration Inventory, and the Zampetakis & Moustakis Scale that assesses entrepreneurial intentions. The FGS did not mention entrepreneurial intentions, and while general engineering and SE share many conceptual ideas, they differ radically in execution. There is no ability to apply a bugfix patch to a badly engineered bridge, for example. Their exploratory factor analysis revealed four main factors: cognitive approaches (the usage of problem-solving tactics, mapping to our CPS domain *creative techniques*), cognitive challenges (the enjoyment of experiences that challenge viewpoints; this somewhat maps to *curiosity*), cognitive preparedness, and impulsivity in problem solving. The last two components are personality-based, which our CPS model does not focus on.

There also exist multiple scales that focus on computational thinking. The Computational Thinking Scale (CTS) of Korkmaz, Çakir, and Özden (2017) contains the following five factors: creativity, cooperativity, algoritmic thinking, critical thinking and problem solving. Although the CTS scale confirms the importance of a few domains that our CPS framework shares, such as the presence of creative and communicative skills, it lacks important aspects from the FGS, such as the influence of constraints and the usage of practical creative techniques. To measure their single "creativity" factor, they selected suitable items from the "How Creative Are You?" scale, the "Problem Solving Scale", the "Cooperative Learning Attitude Scale", "The Scale of California Critical Thinking Tendency", and the "Logical-Mathematical Thinking" scale.

In summary, although self-assessment metrics exist for creativity in general, in an engineering context and in the context of computational thinking, none fully suits our needs. Indeed,

existing scales focus on other aspects of creativity than CPS and do not cover all of the aspects identified by industry experts in the FGS.

11.2 Survey design

For each of the seven domains identified in the FGS, we composed eight questions, based on existing surveys. The full questionnaire set thus contains 56 items. Our intention is to reduce the item set after determining the internal consistency by inspecting Cronbach's α values. Bad questions will be omitted in future versions of the survey. The analysis is described in Section 11.2.

Since we are interested in CPS in the context of SE students, we designed the scale to be as domain-specific as possible, as recommended by Barbot, Hass, and Reiter-Palmon (2019). Therefore, we replace generic questions such as "I like variety" by, e.g.: "I liked varied aspects of the programming assignment".

Assessment tools for creativity can be divided into those measuring creative potential, performance, and achievement (Barbot, 2019). Our CPPST measures *achievements*. It does not gauge creative potential, nor does it measure performance, as Torrance's TTCT or Amabile's (1988) CAT metrics do. The items are self-assessing in nature and ask about the usage of various CPS skills on a recent programming achievement.

All questions were sampled on a Likert-5 scale, which is the most common in creativity research (Kaufman, 2012). Some surveys on which we based our questions use a Likert-6 scale with "not applicable" as the sixth option, but we left this out as all items are deemed relevant. The answer categories are: (1) completely disagree, (2) disagree, (3) neutral, (4) agree, (5) completely agree. Two questions per domain are reverse-coded to check the consistency of the answers. For each survey, questions are served in random order. Students were not able to see the results after finishing the questionnaire.

We now describe the origins of the questions for each of the seven

FGS domains.

Technical knowledge (KNW)

To be able to creatively solve programming problems, one needs to have technical knowledge and to invest in keeping this knowledge up-to-date, by continuously learning and improving oneself. In the FGS, the following themes emerged with relation to technical knowledge: *continuous learning*, *domain models*, *seeking out different inputs*.

To assess these themes, we adapt the following 8 questions from the 14-item lifelong learning measurement scale of Kirby, Knapper, Lamon, and Egnatoff (2010):

1. I have gained little knowledge during the project[3].
2. I learned and applied new practical programming techniques.
3. I have gained insight into the problem domain.
4. The technical aspect of programming appealed to me.
5. I thought about my learning process and how to improve it.
6. I felt uncomfortable with this project because many aspects were unknown[4].
7. I tried to relate the new knowledge to something I know.
8. Thanks to the project I also gained knowledge of other things outside of programming.

[3] Inverted.

[4] Inverted.

Communication (COM)

The FGS stresses that modern SE is a team-based activity. Engaging in discussions helps tremendously in figuring out how to solve difficult programming problems. According to the FGS, communication is therefore an important aspect of creative work in SE.

To assess this, we combined results from the FGS with ideas from the Collaboration Self Assessment Tool (Ofstedal and Dahlberg, 2009). The latter survey is based on the following concepts: (1) *Contributions*, (2) *Quality of Work*, (3) *Time management*, (4) *Team Support*, (5) *Preparedness*, (6) *Problem solving*, (7) *Team Dynamics*, (8) *Interactions with Others*, (9) *Role Flexibility*, and (10) *Reflection*. Since some of these concepts are already

present in the other domains of our CPPST scale, we focus here on communication, collaboration, and team efforts.

Related themes from the FGS are: *rubber ducking* (to explain a problem to oneself or others to gain new insights), *drawing on a whiteboard*, *getting fast feedback*, *working with peers*, *responsibility*. The following questions assess communication in the CPPST:

[5] Inverted.

1. I hardly asked for feedback from my fellow students[5].
2. I visualized the problem on a whiteboard or on paper.
3. I regularly asked feedback from my teachers.
4. I supported my teammates by helping them with their tasks.
5. My own tasks were not completed on time so teammates ran into problems with the deadline[6].

[6] Inverted.

6. I supported the ideas and efforts of my teammates.
7. I was so proud of our result that I showed it to everyone.
8. I thoroughly thought suggestions by others through.

Constraints (CTR)

Constraints outline the context of the problem. They keep you from working on it forever, and they keep the solution relevant to the issue at hand. Most software projects are "brownfield" projects, in which an existing code base comes with many external constraints. Thus, both designing and thinking in context of constraints are very important in software engineering.

Biskjaer et al. (2020) recently demonstrated that there is a sweet spot for accelerating (the potential for) creativity with constraints. Too few constraints cause one to lose interest, while too many constraints may cause stress and hamper creative freedom. According to Biskjaer (2013), self-imposed creativity constraints can also play an important role. For our survey, however, we view such self-imposed constraints as a creative technique, and focus here on external constraints.

Themes that emerged from the FGS are: *client-oriented designing*, *efficiency*, *relevance*, *context*, *performance*, *time-constraints*. The following questions assess the ability to effectively handle constraints:

1. I regularly thought about the correctness of my solution.

2. Due to time pressure, I performed less well[7].
3. I tried to make my code as elegant as possible.
4. I tried to identify the constraints of the assignment.
5. I have had the program tested by friends and / or family.
6. There was too much creative freedom for me, so I could not make a good decision[8].
7. Coding on short notice accelerated my learning process.
8. I regularly tested myself and paid attention to its ease of use.

[7] Inverted.

[8] Inverted.

Critical Thinking (CRI)

Critical thinking is the ability to question things, to come up with alternatives, and to judge the trustworthiness of information sources. A classic counterexample in the SE world is mindlessly copy-pasting code snippets from online sources. Sosu (2013) developed and validated a Critical Thinking Disposition Scale. Their scale is split into two main components after an exploratory factor analysis: (A) Critical Openness and (B) Reflective Scepticism. We adapted an equal number of items from both components to our domain.

Themes that emerged from the FGS are: *coming up with alternatives*, *asking "why?"*. The following questions assess the ability to think critically in the CPPST:

1. In discussions about problems, I often suggested alternatives.
2. I regularly carefully weighed up the various options we had.
3. I dared to completely rewrite my code when it didn't go well.
4. I used multiple sources to find out information myself.
5. I didn't think it was important to ask teammates how they implemented something[9].
6. I always check the credibility of the source when I look something up.
7. It was more important that it worked than that I 100% understood why[10].
8. Looking at other projects made me think about my own.

[9] Inverted.

[10] Inverted.

Curiosity (CUR)

As the FGS states, a "hungry" mind, that is motivated to seek out new things, is very important in SE. Themes that emerged in the FGS are: *getting out of the comfort zone*, *motivation*, *the creative urge*, *complexity is fun*, *admiration*. We base our questions about this domain on the Curiosity Index developed by Fulcher (2004):

1. During the project, I got very much out of my comfort zone.
2. Many parts of the project piqued my curiosity.
3. I enjoyed getting involved in many aspects of the project.
4. I enjoyed really immersing myself in some aspects.
5. I was stimulated by the complexity of the project.
6. I felt the urge to implement extras.
7. I have not had any fun while developing the project[11].
8. I had to commit myself to finish the project[12].

[11] Inverted.

[12] Inverted.

Creative State of Mind (MND)

Optimal creative work requires a certain state of mind, which Csikszentmihalyi (1997) calls *flow*. The FGS mentions that people can tell whether colleagues are being creative by looking at body language: Are they happy, and making a lot of jokes? Are they "in the zone"?

Jackson and Marsh (1996) developed the Flow State Scale to measure Optimal Experience, based on the work of Csikszentmihalyi. It contains 36 items divided into nine factors: (1) Challenge-Skill Balance, (2) Action-Awareness Merging, (3) Clear Goals, (4) Unambiguous Feedback, (5) Concentration on Task, (6) Sense of Control, (7) Loss of Self-Consciousness, (8) Transformation of Time, and (9) Autotelic Experience.

Themes that emerged from the FGS are: *focus*, *flow/being in the zone*, *flexibility*, *environment* (*space/social*), *productivity tooling*. The common divisor of these themes and the factors of the Flow State Scale leads to the following questions:

1. I remained focused for a long time on one part of the project.
2. I used productivity tools to focus more on the essentials (e.g. shortcuts, command line tools, ...

3. I found the experience to be very rewarding.
4. Time seemed to fly while working.
5. I did not know enough to meet the high demands of the project[13].
6. Programming went almost automatically.
7. I did not know what exactly I wanted to achieve[14].
8. I was not concerned with what others thought of my code.

[13] Inverted.

[14] Inverted.

Creative Techniques (TCH)

Creative techniques are concrete actions to accelerate CPS. Hunt (2008) describes various creative techniques specifically for software developers. The FGS also mentioned other simple tricks, such as taking a coffee break or deliberately going to the toilet.

We were unable to find an existing scale that measures the use of such techniques. Therefore, we based the questions solely on the following themes from the FGS: *viewing problems from a different angle* (*birds-eye view*), *asking energizer questions*, *brainstorming*, *mapping problems from different domains*, *combining ideas*, and *seeking out input*. The following questions assess these:

1. I always used the same method to solve a problem[15].
2. I used knowledge from another domain to solve something.
3. I combined different ideas to tackle a problem.
4. I deliberately took occasional breaks to let things sink in.
5. I brainstormed with others to come up with new ideas.
6. I took a step back now and then to see things as a whole.
7. In case of problems I let myself be inspired by other projects.
8. I was regularly stuck[16].

[15] Inverted.

[16] Inverted.

Face validity

We believe that the face validity of our CPPST is by design already quite high, since it is based on existing validated scales and the FGS of Chapter 9, in which 33 experts collaborated on the exploration of creativity in SE. Furthermore, the selected items all resemble real-life CPS behavior.

Throughout the initial development of the scale, we received feedback from the department of psychology at the university of Louvain. One of their recommendations was to use eight items per

domain, instead of five as we had originally planned. Once all of their feedback had been addressed, we administered the survey to a small set of volunteering second-year SE students ($n = 9$), which led us to reformulate some sentences that were hard to understand. The survey was also checked by a number of the FGS participants ($n = 5$), which resulted in rewriting six items and changing the Likert label for value 3 from "more or less" to "neutral". The approval of the items by these members of the FGS provides further evidence for the face validity of the survey.

Survey analysis

We wanted the data to represent a broad range of SE students. Therefore, we surveyed both first-year ($n = 141$) and last-year ($n = 128$) engineering students. The test was administered as close as possible to a project deadline as part of a programming course. The survey was not obligatory, but students were strongly encouraged to participate. Because student demographics in the surveyed group were not very heterogeneous and Miller (2014a) earlier found no significant difference in self-assessment creativity scores based on participant gender, ethnicity or year in school, we decided not to consider demographics as a separate discriminant.

We follow reliability and validity analysis procedures from Miller (2014a) and Korkmaz et al. (2017). However, we will be conservative when deciding whether to drop an item from the scale when factor analysis shows it to be loaded on multiple factors. Since the CPS domains are heavily intertwined, as also confirmed by the FGS, we believe that items loaded on multiple factors cannot be avoided entirely, without losing key information. We will therefore be hesitant to do so, especially if the Kaiser-Meyer-Olkin statistic that indicates the factorability of the item set is high enough (Kaiser, 1974).

For the reliability analysis, we calculated several Cronbach's α values: an initial α for the entire survey, a separate α after a number of items were dropped, and α values for each discovered factor. The items that were dropped were those with doubtful or worse item-total correlations (R_{it}), in accordance with the rules of thumb of Ebel and Frisbie (1972): poor ($R_{it} < .20$), doubtful ($.21 < R_{it} <$

.29), good ($.30 < R_{it} < .39$), and very good ($R_{it} > .40$).

Construct validity is usually investigated with the help of various factor analysis methods (Ebel and Frisbie, 1972). Since our survey is based on the seven domains of the FGS, we first perform a Confirmatory Factor Analysis (CFA) for these domains. This will answer the question of whether the original seven domains can be considered independent factors for our survey. If the answer is no, we will perform an Exploratory Factor Analysis (EFA) to discover more appropriate factors.

11.3 Results & discussion

In total, 269 out of 336 (80%) students participated. On average, the survey took 9.5 minutes. Overall, an initial reliability analysis resulted in a good global α of .89. However, when we look at each domain separately, several domains contained a number of items that have a low item-total correlation, according to the rules of Ebel and Frisbie (1972). After removing the doubtful R_{it} correlations, reducing the number of items from the initial 56 to 37, the global α increased to .91.

The Kaiser-Meyer-Olkin statistic for the 37 remaining items is .90, a value that Kaiser calls "marvelous" for the factorability (Kaiser, 1974). This is also a promising indication for sound construct validity.

We then executed a confirmatory factor analysis with the seven FGS domains as factors. The Comparative Fit Index (CFI) is .76. The Root Mean Square Error of Approximation (RMSEA) is .066, for which the P-value of $RMSEA \leq 0.05$ was zero. According to Kline, an acceptable conformity should report values between the ranges $.06 \leq RMSEA \leq .08$ and $.90 \leq CFI \leq .96$ (Kline, 2015). The bad CFI indicates that the seven-factor model is not a great fit for the data and that, as may have been expected, the FGS domains are not sufficiently independent factors. Therefore, we shifted from confirmatory to exploratory analysis.

A parallel analysis suggests to use between three to five factors, as visible in the scree plot in Figure 11.1. Five- and four-factor EFAs

result in unsatisfactory component matrix distributions, while a three-factor EFA with an oblique rotation reports a CFI of `.91` and a RMSEA of `.06`. We use the oblique rotation because we recognize that there is likely to be some correlation between students' latent subject matter preference factors. Table 11.1 shows the three-factor loadings with principal axis factoring.

Figure 11.1: Scree plot (eigenvalues according to the factors).

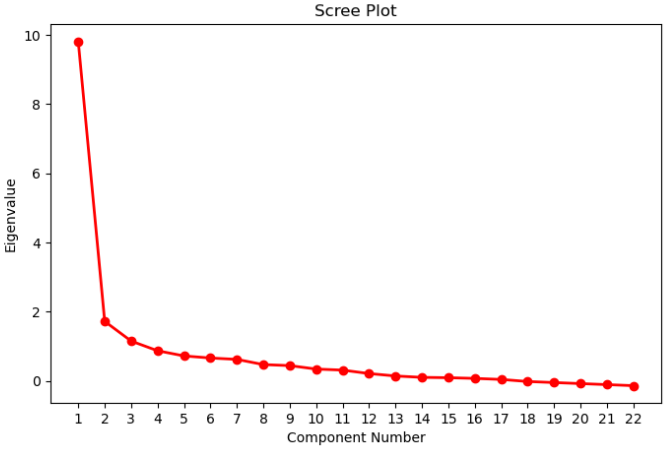


Table 11.1: Rotated Component Matrix: reduced 32-item CPPST set. Loadings below .30 (CRI7, TCH2, TCH5, CTR1, CTR2) are omitted for brevity.

Item	F1	F2	F3
CUR2	.75		
CUR3	.51		
CUR4	.70		
CUR5	.59		
CUR6	.56		
MND3	.71		
MND4	.64		
TCH3	.44		
TCH6	.34		
KNW1	.68		
KNW2	.52		
KNW3	.58		
KNW4	.46	.41	
KNW7	.35		
KNW8	.40		
COM7	.50		
CTR7	.49		
CUR7	.35	.47	
CUR8		.32	
MND5		.67	
MND6		.67	
TCH8		.76	
KNW6		.57	
MND7			.42
COM4			.50
COM5			.42
COM8			.51
CTR3			.33
CTR8			.42
CRI1			.47
CRI2			.31
CRI3			.36

As could be expected from the lower R_{it} correlations in Table 11.3 for domains such as *critical thinking* (CRI) and *creative techniques* (TCH), items related to those domains result in low loading values in Table 11.1. For instance, only three CRI items remain in the component matrix when employing a cutoff rate of `.30`. A reliability analysis on the three revealed factors results in very good Cronbach's α values: `.89`, `.86` and `.83` for F1, F2, and F3 respectively. Full reliability item statistics, calculated individually per domain, can be found in Table 11.2 and Table 11.3. The items in bold are excluded after inspecting `r.cor`.

In the next subsections, we will interpret the results of the pilot study in context of Creative Problem Solving and the FGS, as well as reflect on possible uses and future work.

item	raw.r	std.r	r.cor	r.drop	mean	sd	invert?
CUR1	0.018	0.0232	**0.007**	0.030	3.3	0.96	
CUR2	0.629	0.6222	0.629	0.599	3.5	0.96	
CUR3	0.603	0.6042	0.605	0.574	3.7	0.87	
CUR4	0.677	0.6693	0.678	0.651	3.8	0.94	
CUR5	0.495	0.5006	0.493	0.459	3.4	0.95	
CUR6	0.554	0.5548	0.550	0.516	3.4	1.07	
CUR7	0.636	0.6171	0.622	0.602	3.6	1.09	✓
CUR8	0.553	0.5388	0.535	0.515	3.2	1.07	✓
MND1	0.257	0.2685	**0.247**	0.217	3.6	0.86	
MND2	0.263	0.2475	**0.224**	0.206	3.0	1.21	
MND3	0.663	0.6576	0.664	0.635	3.7	0.95	
MND4	0.563	0.5560	0.553	0.524	3.6	1.09	
MND5	0.626	0.6105	0.616	0.591	3.2	1.10	✓
MND6	0.479	0.4534	0.450	0.434	2.5	1.13	
MND7	0.419	0.4172	0.403	0.379	3.5	0.95	✓
MND8	0.011	0.0014	**0.040**	0.043	3.0	1.09	
TCH1	0.249	0.2366	**0.281**	0.285	2.9	0.79	✓
TCH2	0.319	0.3194	0.299	0.279	3.4	0.88	
TCH3	0.493	0.5116	0.502	0.467	3.7	0.68	
TCH4	0.156	0.1589	**0.128**	0.109	3.7	0.96	
TCH5	0.298	0.3119	0.290	0.258	3.8	0.87	
TCH6	0.296	0.3136	0.290	0.259	3.6	0.80	
TCH7	0.187	0.1952	**0.171**	0.140	3.5	0.99	
TCH8	0.413	0.3907	0.383	0.366	2.6	1.12	✓

Table 11.2: Reliability item statistics for CUR, MDN, and TCH.

Revealed constructs

The factor analysis in Table 11.1 shows that, except KNW4 and CUR7, the items are very evenly distributed across the three factors, which are deemed reliable thanks to the high α values. Factor F1 contains most Curiosity items, almost all Knowledge, and Creative Techniques items, and some Creative State of Mind items. These 17 items make up for the majority of the question pool. It is clear that this group is knowledge oriented. Furthermore, knowledge and curiosity-related questions show particularly high loadings. We call this construct **Ability**. To be able to solve a programming problem creatively, one first needs to be able to program. Next to programming skills, knowledge of creative techniques that speed up the process are a big help. Curiosity may be a key motivation for investing the time and effort in acquiring all of these skills. KNW4, "the technical aspect of programming appealed to me", is loaded on both F1 and F2. Perhaps some

Table 11.3: Reliability item statistics for KNW, COM, CTR, and CRI.

item	raw.r	std.r	r.cor	r.drop	mean	sd	invert?
KNW1	0.582	0.5903	0.593	0.555	4.0	0.80	✓
KNW2	0.478	0.4923	0.483	0.450	3.9	0.71	
KNW3	0.577	0.5868	0.584	0.551	3.7	0.77	
KNW4	0.613	0.6016	0.606	0.578	3.5	1.10	
KNW5	0.242	0.2494	**0.226**	0.199	3.4	0.90	
KNW6	0.574	0.5547	0.555	0.535	3.2	1.14	✓
KNW7	0.373	0.3781	0.360	0.336	3.7	0.83	
KNW8	0.461	0.4429	0.435	0.418	3.1	1.06	
COM1	0.191	0.2063	**0.180**	0.147	3.7	0.91	✓
COM2	0.324	0.3066	**0.286**	0.267	3.2	1.25	
COM3	0.242	0.2319	**0.206**	0.191	2.9	1.08	
COM4	0.464	0.4846	0.475	0.436	4.0	0.70	
COM5	0.372	0.3724	0.359	0.335	4.2	0.87	✓
COM6	0.173	0.2033	**0.181**	0.143	4.1	0.62	
COM7	0.501	0.4885	0.483	0.456	3.3	1.16	
COM8	0.301	0.3333	0.316	0.275	4.0	0.56	
CTR1	0.397	0.4186	0.405	0.369	3.9	0.68	
CTR2	0.463	0.4497	0.438	0.418	3.3	1.12	✓
CTR3	0.491	0.4939	0.487	0.457	3.6	0.89	
CTR4	0.251	0.2583	**0.234**	0.214	3.3	0.79	
CTR5	0.318	0.2941	**0.277**	0.256	2.7	1.35	
CTR6	0.276	0.2680	**0.244**	0.229	3.4	1.01	✓
CTR7	0.457	0.4605	0.450	0.418	3.5	0.96	
CTR8	0.408	0.4215	0.406	0.375	4.0	0.77	
CRI1	0.412	0.4363	0.426	0.382	3.7	0.73	
CRI2	0.401	0.4208	0.405	0.370	3.5	0.73	
CRI3	0.412	0.4142	0.400	0.370	3.8	0.99	
CRI4	0.259	0.2659	**0.244**	0.215	3.7	0.94	
CRI5	0.131	0.1321	**0.099**	0.079	3.9	1.05	✓
CRI6	0.181	0.1834	**0.157**	0.133	3.2	0.98	
CRI7	0.322	0.3154	0.296	0.271	3.1	1.12	✓
CRI8	0.222	0.2310	**0.207**	0.178	3.6	0.93	

students are mesmerized by the programming challenge while others remain indifferent—independent of their technical ability.

Second, factor F2 contains Creative State of Mind items and a number of related items. For instance, TCH8, the highest loaded item, states "I was regularly stuck", which is also related to one's mindset on how to approach a problem. The feeling of being comfortable with unknown aspects (KNW6) is also linked to one's belief in own abilities. CUR7, "I had fun while developing the project", is also loaded on both F1 and F2. This could perhaps indicate that, as Figueiredo, Lopes, and García-Peñalvo (2019) mentioned, while having fun is stimulating to learning to program,

"a lack of success on the (introductory) programming courses, can also be a demotivating factor". A minimum level of ability could increase the coding fun.

As F2 items are related to a way of thinking, we call F2 **Mindset**. It seems related to what Duckworth, Peterson, Matthews, and Kelly (2007) call "Grit" or what Dweck (2008) calls a "Growth Mindset". In a recent study on the relation between mindset in computing students and their study performance, Apiola and Sutinen (2020) found that mindset on computing was growth-oriented, but that mindset on creativity was the most fixed of all scales. This means that students either think they are creative, or that they are not, but they are not open to the idea of nurturing creativity. A tool such as our CPPST may be used to track how students' creativity increases over the course of a curriculum, thereby helping to cure students of this misconception.

Lastly, factor F3 contains almost all communication-related items. Other items, such as "in discussions about problems, I often suggested alternatives" (CRI1) and "I paid attention to its ease of use" (CTR8) also involve a interactive element. Therefore, we call F3 **Interaction**. Social aspects have already been shown to be a critical component of CPS (Amato-Henderson et al., 2011). External suggestions, support, and criticism are important to build creative solutions and to build them quickly. Creative programmers know how and how often to interact, and how to critically interpret responses[17]. Katz (1984) talks about "collective problem solving" and discusses its many advantages over individual problem solving, which seem to translate well into the field of SE.

[17] *The Creative Programmer* (Groeneveld, 2023) contains a plethora of practical examples from the SE world related to communication and interaction. See the conclusion at page 263.

First-year versus last-year students

The development of the CPPST is geared towards higher education students enrolled in a computing study program. We surveyed both first-year and last-year students and analyzed the data as a whole to better represent the entire curriculum. When inspecting the data of first-years and last-years separately, we noticed slight differences in item-total correlation and Cronbach's α values for certain FGS domains. While CUR, MND and KNW items stay the same, many

last-year COM items have lower correlations. For example, COM1 and COM3 are about regularly asking feedback from fellow students and peers. Perhaps graduates feel too confident and think they don't need feedback. Perhaps feedback is not perceived as something that can enhance your own creative ideas, but as something obligatory related to grading.

We argue that, next to these quantitative results, qualitative data can place these differences into context. Still too little is known about the perception of creativity in the minds of SE students. Chapter 10 provides some answers in this direction. However, since only five first-year and five last-year students participated, there is still room for future work. We feel confident about our industry-academia comparison, but a thorough first-year vs last-year comparison is yet to be made.

Summary

Our factor analysis yields three major components for CPS in SE: *Ability*, *Mindset*, and *Interaction*. These factors somewhat overlap with the aforementioned engineering scale of Amato-Henderson et al. (2011). They discovered four factors: cognitive approaches (not unlike our factor *Ability*), cognitive challenges and cognitive preparedness (somewhat overlapping with our factor *Mindset*), and impulsivity. The major difference is the absence of any interactive component, although they also do mention the importance of social aspects throughout their paper. In general, many creativity assessment toolkits lack an interactive component. Another example is the work of Denson, Buelin, Lammi, and D'Amico (2015), where creative design is composed of technical strength, aesthetic appeal, and originality. It is clear to us that our first component, *Ability*, can easily be found in other metrics. Some metrics take *Mindset* into consideration, and almost none *Interaction*.

11.4 Limitations

To reduce threats to validity as much a possible, we closely followed the recommendations for creative assessment outlined by Barbot et al. (2019). For example, they indicate that measuring creative achievement is perhaps the most objective way to assess creativity. We therefore formulated the questions in the context of a concrete achievement (a programming project), and administered the survey as close as possible to the project deadline.

In general, self-rating is bound to introduce some form of bias. People are usually inaccurate in assessing their own "generic creativity", as stated by Barbot et al. We therefore chose to avoid the word *creativity* altogether in our survey. Instead, we opted for descriptive, concrete questions, related to programming as much as possible. This also sidesteps potential issues due to a fixed mindset on creativity as mentioned by Apiola and Sutinen (2020).

We believe that the results of this pilot study show promise. The Kaiser-Meyer-Olkin statistic indicates a high factorability, and the Comparative Fit Index and Cronbach's α reliability values confirm that the three-factor model is a good fit. The three major constructs identified by the exploratory factor analysis are interesting and explainable, albeit in need of more qualitative context to better understand the perception of students' creativity.

11.5 Conclusion

In this chapter, we explored the possibility for undergraduate software developers to self-assess their CPS skills through a survey that encompasses seven dimensions of creativity, based on existing validated scales and conducted focus groups. The Creative Programming Problem Solving Test or CPPST is a survey geared specifically towards computing education, as we found that existing creativity metrics are either too broad or too narrow, and do not match well with expectations of software engineering industry experts.

Principal axis factor analysis grouped the focus group dimensions into three overarching constructs: (1) *Ability*, (2) *Mindset*, and (3) *Interaction*. We believe that the found constructs

can be used as important building blocks for amplifying students' creativity in computing education. Furthermore, this work contributes to computing education research by bringing research on creativity in the field of cognitive psychology closer to the field of computing education.

This chapter concludes Part III, which sought to understand what exactly it means to be a creative software engineer. We now have the knowledge and tools to be able to move on to Part IV, where we will zoom in on overarching RQ3—how to improve SE education—by exploring how this test can be used to gauge the creativity levels of a student assignment.

Part IV

Amplifying Creative Skills

In this part, we try to answer the **third RQ** of this work:

PhD Research Question 3

How can we improve upon education to reduce the industry-academia skill gap?

Again, in order to answer this question, we conducted several separate works, which have been peer reviewed and published. First, in Chapter 12, we approach this question by increasing student motivation and engagement in creative open assignments (Groeneveld, Martin, Poncelet, and Aerts, 2022b).

Second, in Chapter 13, we investigate the effects of increased creativity through open assignments on the code quality of first-year engineering students (Groeneveld et al., 2022b).

Third, an interdisciplinary approach was taken to help boost the creative problem-solving skills of students, where the CPPST from Chapter 11 was deployed to zoom in on the three specific parts of creativity (ability, mindset, interaction) (Duyver, Groeneveld, and Aerts, in press).

Each of these chapters attempts to improve education by reducing the skill gap that was introduced in Part I and Part II. The first two studies were conducted in parallel with the exploration of the concept of creativity as presented in Part III, which explains why only the last chapter of this part builds upon the CPPST.

12 Creativity & Motivation

Publication details

This chapter has been published in the *Proceedings of the 48th SEFI Annual Conference 2020* as *Engaging Software Engineering Students in Grading: The effects of peer assessment on self-evaluation, motivation, and study time* (Groeneveld, Vennekens, and Aerts, 2020c).
Its main contribution is promoting peer grading as a tool to help students understand and evaluate their own project, boosts their motivation to work on it, and increase the relation between motivation and time spent on the project.
In addition, a poster that presents these student projects was published in the *Proceeding of the 51th ACM Technical Symposium on Computer Science Education* as *Sparking Creativity with the Game Boy Advance* (Groeneveld and Aerts, 2020).

Open assignments are one of the most popular teaching methods to bring creativity to the classroom, as mentioned in Chapter 3. As part of a curriculum revision at our local faculty of Engineering Technology, we have developed a course to teach object-oriented software design in C++ using the Game Boy Advance (GBA). The 2001 GBA gaming handheld from Nintendo lets students develop software on low-end embedded hardware, and of course the nature of the machine motivates students compared to the more conventional academic assignments.

We will first describe the teaching methodology for the *Software Design in C++* course as mentioned in the published poster (Section 12.1). As an extension of that poster, we were also interested in seeing whether students could be even further motivated by introducing peer-grading in the same GBA-centered course. Section 12.2 will cover the findings of our peer grading experiment.

12.1 Sparking creativity with the GBA

While completely reworking the *Software Design in C++* course in 2018, we focused on real-life, embedded system experiences to trigger the creative skills of students. In total, between 30 and 35 engineering students are enrolled each year in this obligatory course[1]. As previous studies have shown (Finlayson, 2017; Papastergiou, 2009), the introduction of the GBA is a great way to trigger the interest of students. In context of the seven dimensions of creative problem solving as revealed in Chapter 9, we wondered: *how can we stimulate all dimensions of student creativity*, within the limits of a programming course?

[1] The course, consisting of 4 ECTS points, was part of a bridging module for professional bachelors wanting to pursue a master's degree and for third-year academic bachelor Electronics/ICT students. The course got scrapped in academic year 2022–23 and was superseded by a 6 ECTS *Operating Systems in C* course that also focuses on low-level hardware interaction.

Besides the teaching of technical *expertise*, we also focused our exercises on stimulating *creative skills*. For example, working with constraints (e.g. splitting sprites to conserve memory space) is very important in a limited environment such as the GBA. To minimize technical frustrations, we built an object-oriented sprite engine that students can use in their project. The engine[2] hides low-level plumbing code in favor of readable high-level methods.

[2] The open source engine is available at `https://git.brainbaking.com/wgroeneveld/gba-sprite-engine/`.

An overview of the C++ course and a video showcasing the work of students is available at `https://brainbaking.com/post/2019/04/teaching-oo-with-gba/`.

During the first year after reworking the course, we collected initial feedback from 24 students using a short survey. Only 4% of the students admitted not to be motivated. Main motivators were:

1. Doing something else than working with normal micro-controllers;
2. Being able to make a game on the GBA;

3. The ability to play it on actual hardware;
4. Immediate graphical feedback;
5. Creativity of assignment due to freedom;
6. Uniqueness; and
7. Nostalgia.

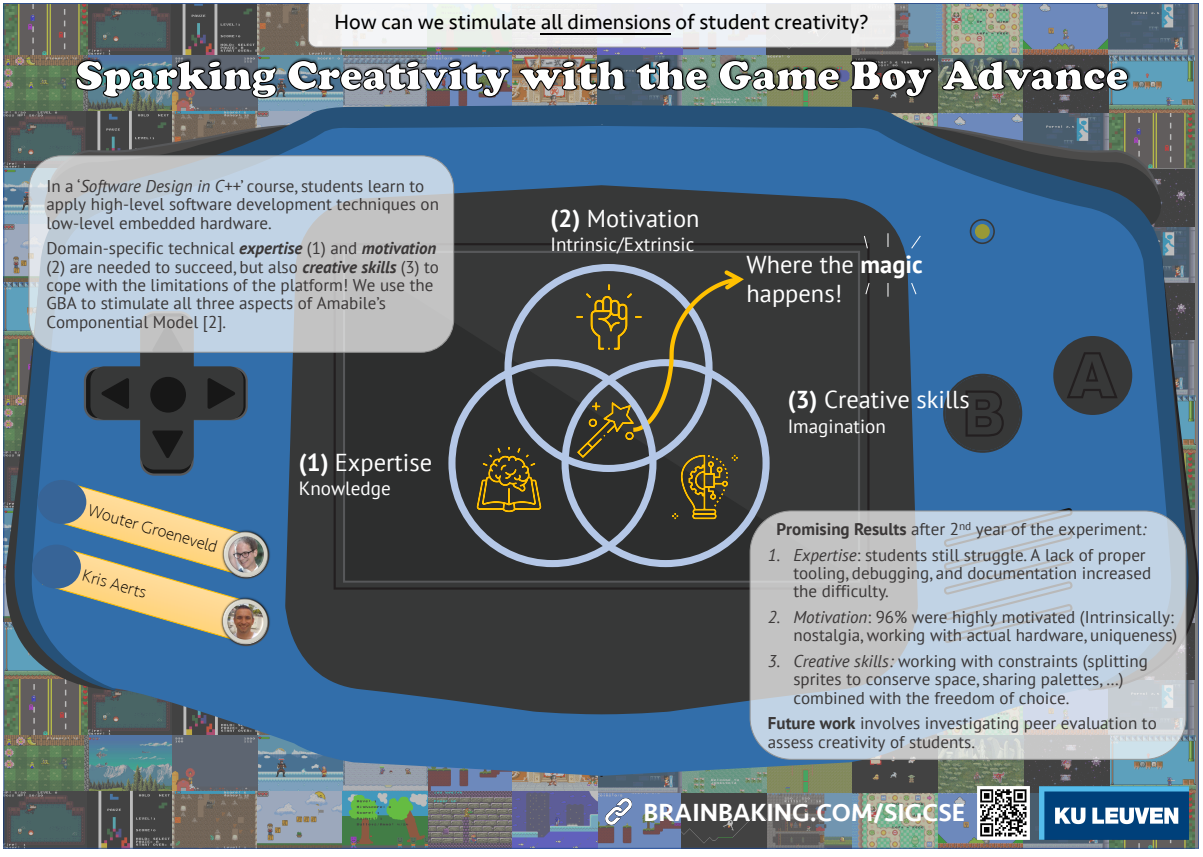


Figure 12.1: The GBA course poster as presented at SIGCSE 2020.

The main demotivators were:

1. Unfamiliarity with tools;
2. Debugging difficulties;
3. A high initial general difficulty;
4. Too ambitious projects.

We used Amabile's (1982) CAT to evaluate the creativity of the end product[3]. Instructors rated each game on different criteria, each with their own weight: design (`0.5`), clean code (`0.5`), usage of conventions (`0.1`), complexity (`0.3`), originality (`0.3`), UI (`0.2`). This covers both the technical and creative skills dimensions of creativity.

In addition to the survey, we asked students to record their time spent on the project. In the next section, we will introduce a peer-evaluation system that enables students to assess the creativity of others using an evaluation rubric system[4].

[3] An in-depth discussion of the applied CAT methodology is available in Chapter 13. Due to space limitations, this was not part of the poster.

[4] This research was carried out in 2019–2020, while the CPPST was developed in 2022. The rubric evaluates more than simply creativity, but it would certainly be possible to use the CPPST to peer-evaluate the creativity criteria.

12.2 *Engaging students in grading*

PEER ASSESSMENT HAS OFTEN been used to enhance non-technical skills of students in higher education (Fallows and Chandramohan, 2001; Sluijsmans, Dochy, and Moerkerke, 1998). However, its effects on the learning abilities of students are often overlooked. In this section, we explore self-evaluation, motivation, and study time of students in relation to peer assessment. The development of self-evaluative skills and the awareness that lifelong learning is a necessity for an engineer are our main goals. We are therefore interested in the following research questions, for which there is currently only sporadic evidence:

Chapter Research Questions

1. *What are the effects of peer assessment on self-evaluation?*
2. *What are the effects of peer assessment on motivation and study time?*

We first provide a brief overview of related work before clarifying the process we have followed to collect data spread across two academic years. Then, we present and discuss our findings and list possible limitations.

Related work

As stated by Sitthiworachart and Joy (2004):

The use of peer assessment is claimed to enhance students' evaluative capacities, which improve the quality of their subsequent work.

Several papers, even from more than 20 years ago, argued that using self and peer assessment increases the students learning capabilities (Sluijsmans et al., 1998). Peer assessment is indeed "*as much about learning as assessment*" (Sitthiworachart and Joy, 2004). However, many papers lack a good body of empirical data to back up these claims, except for qualitative surveys and self-reports.

It is important to note that in the context of this work, we are not focusing on the validity of peer assessment as a tool to completely replace the grading of teachers. According to Sajjadi, Alamgir, and von Luxburg (2016), intricate models that would correct any bias or variance in students' grading seems to be rather ineffective. Researchers do not completely agree on the validity of peer grading: some find it to be adequate (Sadler and Good, 2006; Stefani, 1998), while others find it very variable (Falchikov and Goldfinch, 2000; Gracias and Garcia, 2013). As our intention is to introduce peer assessment primarily as a learning tool for students, teacher assessment was not completely removed. The next subsection explains this grading system in greater detail.

Methodology

In the *Software Design in C++* course for third-year bachelor students Electronics/ICT at our local faculty of Engineering Technology, we asked students to grade themselves, fill in a follow-up survey on motivation, and to keep track of their study time on the project of the course. In the second academic year, peer assessment was introduced.

The first academic year, 2018–19 ($n = 25$ students), served as a baseline, while the introduction of peer assessment in the second academic year, 2019–20 ($n = 27$ students), served as the study group. As part of the evaluation process for this course, students are required to complete an integrated assignment presented as a project, which accounts for 50% of the total score (divided into 27% peer grades, 23% teacher grades). Project assignments are very open-ended: students can choose to implement anything they like, as long as the following requirements are met:

- The target programming language is `C++11`, and the target platform is the GBA. As expected, every student opted for the creation of some kind of 2D game.
- A GBA sprite engine was custom built for this course to reduce the technical challenges of working with a low-level embedded hardware system such as the GBA, as explained in Section 12.1. Students were required to start from this engine.

An evaluation rubric was adopted from Gracias and Garcia (2013) to make both the self and peer evaluation process easier for students. This rubric was also employed by the teaching staff in both academic years. Since the Software Design course is a programming course, different non-technical and technical criteria were needed to evaluate different aspects of student projects. Table 12.1 summarizes this rubric and provides examples of low and high scores. Cardinal rating was employed by providing a score for each criterion between `0` to `5`, after which weights are assigned and a global score on `20` is calculated.

Weight	Criteria	Examples
`0.4`	**Code Design**	*How well-designed is your code?* High score: clearly recognizable objects, represented in domain model, separation of concerns. Low score: All code in single object, unclear what does what, barely or no model.
`0.5`	**Clean Code**	*How readable is your code?* High score: use of understandable variables, methods, classes. Low score: Too much re-reading is needed to see what is happening.
`0.4`	**Complexity**	*How difficult did you make it?* High score: chosen for a challenge instead of a simple implementation. Low score: path with smallest resistance taken, the bar set too low.
`0.3`	**Creativity**	*How original and creative is your idea?* High score: implemented an original idea instead of a clone of a default 2D platformer. Low score: opted for a less inspiring design, everything is based on existing work
`0.2`	**GBA UI**	*How elaborate is the presentation of your game?* High score: All UI/Sound techniques applied well: sprites, scrolling BG, ... Low score: little to no animation/backgrounds, monotonous design.

Table 12.1: The qualitative evaluation rubric used to grade the projects.

This rubric was made public in the very first lesson of the course, so that students could take this into account during the design and development of their project. Students were encouraged to work together in groups of two or three to further reduce the complexity and stress of cross-compiling for the GBA.

Peer assessment was obligatory and took place a day after the submission deadline. Every group had exactly five minutes to

demonstrate their project to their fellow students and the teaching staff. We recommended students to spend at least two minutes explaining the overall structure of the code, since two criteria are technical and evaluate the code quality. After the peer assessments, students were asked to evaluate their own project using the very same rubric.

To measure accuracy and motivation, a short follow-up survey was filled in after the self assessment assignment, in which students had to answer the following questions:

1. How easy was it for you to evaluate yourself? (Likert scale, 1–5)
2. How accurate do you think your evaluation is? (Likert scale, 1–5)
3. How motivated were you while working on the project? (Likert scale, 1–5)
4. What motivated you? (Open-ended question)
5. What demotivated you? (Open-ended question)

To measure study time, students were required to keep track of the amount of time spent on working on the project. A template `csv` file was provided for them to fill in, noting the date and the number of hours spent working on the project that day. A simple `csv` file was used to reduce the administrative load as much as possible, while still maintaining a high accuracy rate. A data correction step was necessary as many students made mistakes in noting the year (1018 instead of 2018, 2018 instead of 2019 in January, ...).

Data was not collected anonymously since we wanted to compare self-grading with the actual grading marks. However, since the authors are also the teachers of the course, the data was made anonymous during the analysis of this research to prevent any bias that may result from knowing student names.

Results & discussion

The collected data from two years was analyzed to help us estimate the effects of peer assessment on multiple variables. We will discuss the results based on the two research questions from Section 12.2.

Q1—The effects of peer assessment on self-evaluation

Correlations between self assessment and teacher assessment are summarized in Table 12.2, and visualized as scatter charts in Figure 12.2. The Pearson correlation coefficient was calculated using the individual grades. Creativity was added to the evaluation rubric in the second year.

	Design		Clean Code		Complexity		UI		Creativity	
	Stu.	Tea.	Stu.	Tea.	Stu.	Tea.	Stu.	Tea.	Stu.	Tea.
Mean	3.17	3.71	3.54	3.08	3.13	3.04	3.21	3.50	/	/
s.d.	1.010	1.197	0.658	0.881	0.850	1.233	1.062	1.142	/	/
Corr.	**0.331**		**-0.231**		**0.202**		**0.376**		/	
Mean	4.10	3.33	3.98	3.00	4.13	2.96	2.67	3.19	3,67	3.31
s.d.	0.775	0.582	0.608	1.158	0.558	1.523	0.836	1.184	0.679	0.664
Corr.	**0.105**		**0.341**		**0.666**		**0.713**		**0.528**	

Table 12.2: Comparison of self (Stu.) and teacher assessment (Tea.): mean, standard deviastion, and correlation values for each criterion in the evaluation rubric.

When comparing the means between different years, we can clearly see that students who participated in peer grading generally graded themselves higher. This bias towards higher student gradings, as reported by Gracias and Garcia (2013), is also visible in Figure 12.2, where triangles evolve towards the left side of the centerline. Furthermore, as the correlation increases, the deviation decreases, revealing a higher level of agreement among the scores given by different students to the same project.

Students in the second year report that they find it easier to self-evaluate after peer evaluation. Means from the Likert scales (1–5) are displayed in Table 12.3. This signifies that students get more comfortable evaluating projects during the peer grading process.

Table 12.3: Self-reported means for ease of grading and accuracy. 1 (hard/inaccurate) - 5 (easy/accurate).

Year	How easy to grade?	How accurate? (reported)
2018-19	3.25	3.25
2019-20	3.42	3.62

Evaluating *Design* and *Clean Code* Students reported in the follow-up survey that it was very hard for them to objectively evaluate the criteria *Design* and *Clean Code*. One needs a firm grasp of programming techniques and a lot of experience in reading and

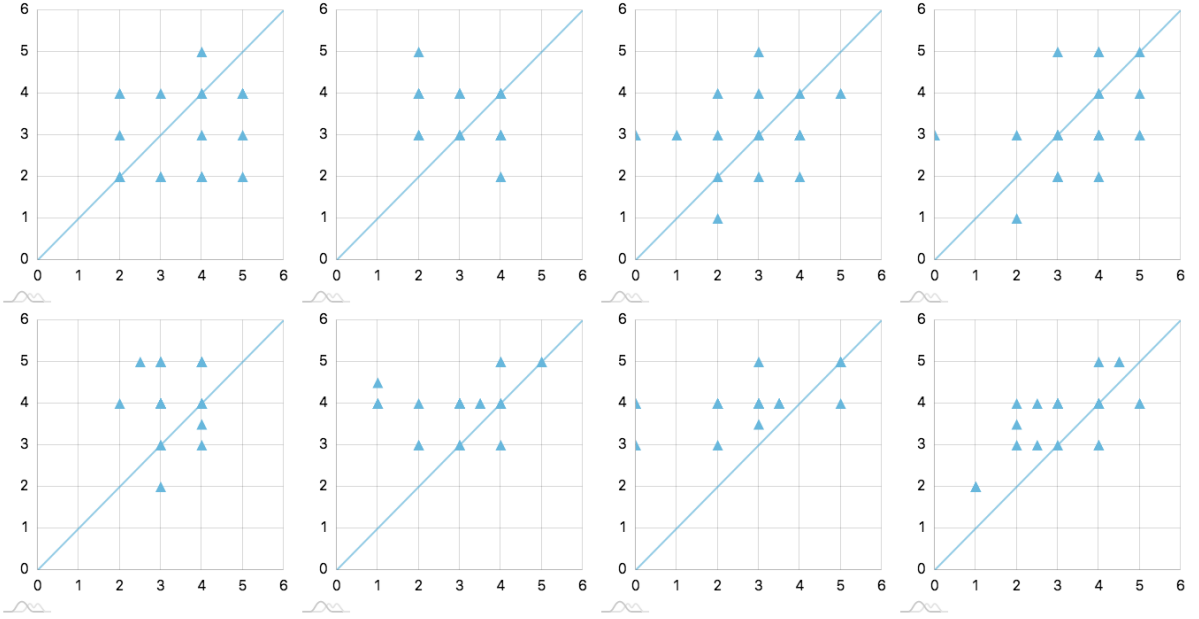


Figure 12.2: Scatter plots of self-evaluation (Y-axis) related to teacher-evaluation (X-axis) for each criterion. The closer to the centerline, the higher the correlation and the better the student's estimate. Left to right: Design, Clean Code, Complexity, UI. (Above: year 2018-19. Below: year 2019-20.)

evaluating code to be able to make a correct judgment. A few lab sessions are effectively dedicated to introducing students to these concepts. However, as they are still inexperienced, it is indeed very difficult to make a good estimate - especially within the short time-frame of the presentation. No code was assessed in advance.

There is no strong relation between self and teacher grading in *Design* (`0.331` to `0.105`). However, the negative correlation in *Clean Code* evolves into a weak relation (`-0.231` to `0.341`), indicating that students at least have some idea on what is "clean" and what is not[5], after they have seen more code examples from their peers. This is an important lesson for us, as in the coming years more examples should be introduced in the labs, where room for discussions can lead to a better understanding of the concept.

Since the evaluation of both technical criteria relies on the assessment of source code text, other assessment methods such as comparative judgement could further increase the reliability of self and peer grades, as discovered by Goossens and De Maeyer (2017).

[5] In Chapter 13, we delve deeper into the possible relationship between clean code and creativity.



Figure 12.3: A selection of the created GBA games.

Evaluating *Complexity* and *UI* A strong increase in the correlation between self and teacher grading was registered for both *Complexity* (0.202 to 0.666) and *UI* (0.376 to 0.713), as also visible in Figure x. The more examples students see and have to grade, the better they can estimate their own project on these criteria. Both *Complexity* and *UI* can be graded by looking at the end product (the GBA game) rather than the source code (C++ files). It is easier to evaluate whether your own game is less or more complex than others if you saw what your peers made. The same holds true for *UI*, where the biggest correlation was noted.

Students frequently applauded others for their ingenuity and resourcefulness. For example, multiple students were asking technical questions on how a group implemented dynamic background switching of their 'Mr. Driller' game. We felt they got inspired simply by taking a good look at the work of others. When looking at the correlations, we can conclude that this also triggered an introspective process that benefits the learning process of students.

Evaluating *Creativity* Every year, students report that the single biggest motivator for them is the possibility to be creative with the GBA. Therefore, the *Creativity* criterion was added during the second year. No comparisons can be made. However, a weak correlation was registered after the peer grading process (0.528).

The problem with creativity is that, as discussed in Chapter 7, it can be interpreted in multiple ways, even if examples of good and bad grades have been provided in the rubric.

We are positive that this correlation can be easily increased provided that a clear definition of creativity is given. The projects are only one dimension of the 4P Creativity Model: the Product (Gruszka and Tang, 2017). The Process dimension is something we already touch upon in the labs by introducing creative techniques to cope with the hardware limitations of the older GBA system. Our aim for the future is to better educate students on what it means to be creative. We are convinced that peer assessment can play a big role in the creativity of students, as also noted by Papaleontiou-Louca, Varnava-Marouchou, Mihai, and Konis (2014):

> [...] *As a guide for creativity, in order to promote creativity as product, with a view at the practical site of teaching, teachers should:* [...] *encourage self and peer assessment and evaluation.*

Q2—The effects of peer assessment on motivation and study time

Self-reported motivation Students report an average motivation of `3.58` in year 1, and of `3.96` in year 2 (Likert scale, 1-5). It is hard to say that this increase in motivation is due to the introduction of peer grading alone. Therefore, we also asked students to write down their motivators and demotivators, of which a selection of the answers is presented in Table 12.4.

It is perhaps striking to see that none of the students in the second year actually mentioned the peer evaluation process itself. However, that does not mean that their motivation was not influenced by the introduction of it. A classic mix of intrinsic ("*fun*") and extrinsic ("*good grades*") motivation was reported, as expected, with a clear bias towards intrinsic motivation. Of course, the fact that one could program on a gaming hardware device is one of the biggest reasons to be highly motivated. The main demotivators are related to the technical challenges of the project assignment.

Study time measurement breakdown Figure 12.4 visualizes the total invested time per day for all students during the given

Stud.	Motivator (+)	Demotivator (-)
1	Creating something out of nothing	Things that did not work on the first few tries such as backgrounds and pointers
2	Great freedom to choose what is made	reverse engineering the GBA hardware
3	Passing the course	GBA memory issues
4	Using own inspiration, open nature of the project	no debugging
5	Creating a game, gaming to test it	Background difficulties
6	Being creative and looking for solutions	Bugs
7	GBA Nostalgia	Exams
8	It is something else than a default assignment	Limited memory capabilities, complexity
9	Seeing graphical progress when developing the game	Too much new concepts to learn

Table 12.4: A sample of the reported motivators and demotivators for the GBA project of the course, throughout both academic years.

academic years, divided into three parts. The biggest peak in the first part, mid December, is the last lab where students are allowed to work freely on their project. Most small dips are Sundays. During the second part, the Christmas holiday period, students clearly start worrying about their project and gradually invest more time.

In the third part, a race towards the submission deadline begins, ending with almost 100 collective hours on a single day, averaging on 4 hours per student.

No significant change in workload is visible between the two years, except for the overall increase in the last race for the deadline. However, the average amount of hours per student decreases slightly, being 39.63 in the first year and 36.61 in the second. This leads us to conclude that having to evaluate fellow students does not seem to change the way students take on a project in general. They might overly rely on the good-will of others to give high grades. The usage of the peer evaluation system was stressed multiple times throughout the semester to make sure students remember how the evaluation of the course works, with a small negative effect on study time.

Link between motivation and study time Table 12.5 displays the correlation matrices for dependent variables time spent, motivation, and total grade. We suspected that motivation and study time investment would be closely linked together. However, without peer grading, no relation could be determined between

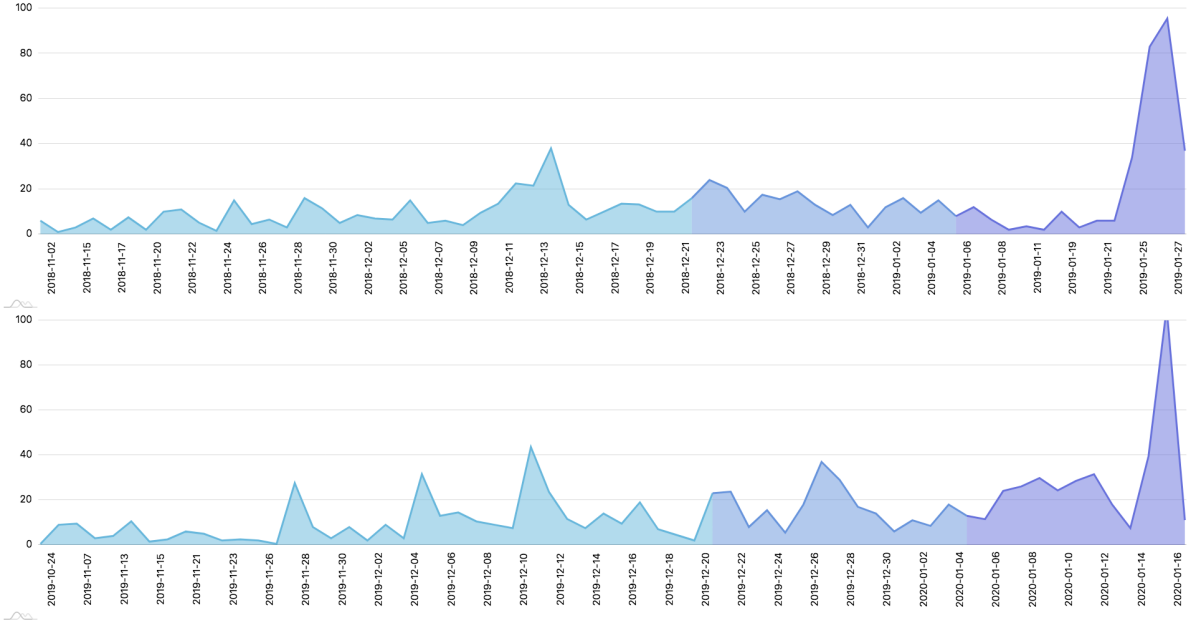


Figure 12.4: A normalized time graph of total invested hours, broken down in three color-coded parts: before Christmas break, Christmas break, before deadline. (Above: year 2018-19. Below: year 2019-20.)

time spent and motivation. The raw data reveals students with high grades that made little investment into the project, and students with low grades that made big investments. Averages on total grades are `13.73` for year 2018–19, and `13.29` for year 2019–20.

Year		Time spent	Motivation	Total Grade
2018–19				
	Time Spent	-		
	Motivation	**0.005**	-	
	Total Grade	**0.394**	**-0.367**	-
2019–20				
	Time Spent	-		
	Motivation	**0.331**	-	
	Total Grade	**0.292**	**0.423**	-

Table 12.5: Correlation matrices for year 2018–19 and 2019–2020.

No strong association could be derived with a correlation higher than `0.5`. However, note again the big increase in correlations on motivation linked to time spent and total grade. Coupled with the

fact that student's self-reported motivation increases overall, the higher correlation suggests that the peer evaluation caused at least the motivated students to invest more time in the project. In a study carried out by Krapp (1999), the distinction was made between intrinsic motivation called interest, and extrinsic motivation. Krapp discovered, as one would expect, that interest is more highly related to study time than extrinsic motivation. Our follow-up survey reveals the presence of both motivator types, which makes it harder get a distinct correlation.

12.3 Limitations

The biggest threat to the validity of this research is the limited scale of execution. It is not the student group size that is a concern, but the longitude of the research. As we measured only two years to date, more years could mitigate possible shortcomings in our study. However, the strongly deviating correlations presented in Section 12.2 are enough to substantiate the results. Existing research further supports these results. Our intention is to continue with the data collection and the peer grading process in the coming years.

The peer evaluation process itself required students to fill in a form on paper. It is not impossible for students to copy grades of their neighbors if they face difficulties while assigning a grade. However, this possible bias and/or variance was mitigated by still taking teacher grades into account. We did not aim for perfect peer grades, but rather for the positive effects of the grading process itself. Extreme positive biases, for example bulk grading or complete random grading, were not encountered.

12.4 Conclusion

Peer assessment drastically improves the quality of self assessment. Gracias and Garcia (2013) ask "*Can we trust peer grading?*" and conclude with mixed results. We would like to

change the question to "*Can we trust the positive effects of peer grading?*". The answer is undoubtedly yes. Students are more motivated and can more accurately evaluate their own work, even if there is no bigger quantifiable time investment measured. In contrast to a report by Sherrard, Raafat, and Weaver (1994), our students did not express any reservations regarding the fairness of the peer grading system. Perhaps this is because teacher grades are still taken into account.

Future work involves providing a clearer explanation of and focus on the evaluation rubric, especially the more technical criteria. The combination of peer assessment and creativity deserves special attention. After seeing the time investment interval from Figure 12.4, we also ask ourselves how to distribute this more proportionally. For instance, earlier peer assessments at multiple intervals during the development of the projects could help. Other educators had success using this "lean" practice of providing earlier, and more frequent, feedback (Chatley and Field, 2017).

The peer assessments presented in this chapter could be viewed as a simplified version of Amabile's (1982) Consensual Assessment Technique. The evaluation rubric used to score the projects contain three distinct *code quality* criterion: *code design*, *clean code*, and *complexity*. In the next chapter, we will zoom in on the possible relationship between creativity and code quality with the help of a static code analysis tool and a more thorough iteration of the CAT tool.

13 Creativity & Code Quality

Publication details

This chapter has been published in the *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education* as *Are Undergraduate Creative Coders Clean Coders? A Correlation Study* (Groeneveld et al., 2022b).
Its main contribution is the discovery of a negative relationship between the perceived creativity and the code quality of first-year student projects, thereby calling for the necessary attention to clean code principles as creativity is promoted in higher education.

Many computing courses introduce a layer of creative freedom by employing open project assignments—assignments akin to the GBA projects from Chapter 12. In this chapter, we are particularly interested in the quality of the submitted projects in relation to the creativity that students showcase when tackling these open assignments.

To investigate whether there is a relation between creativity and clean code in Computer Science 1 (CS1) student projects, we have analyzed 110 projects from two academic years. Seven judges were recruited that evaluated the creativity based on Amabile's (1982) Consensual Assessment Technique, while the PMD tool was used to explore code quality issues in the Java projects.

Specifically, we wondered the following:

Chapter Research Questions

1. *Is there a relationship between creativity and code quality in CS1 student projects?*
2. *Are certain code quality aspects more related to creativity than others?*

By answering our research questions, we can better direct our CS1 teaching focus, thereby potentially increasing both students' creativity and their awareness of clean code principles.

In Section 13.1, we first explore the concept of code quality and clean code in existing work, before explaining the employed methodology in Section 13.2 and describing and discussing the results in Section 13.3.

13.1 Existing work

In a 2018 paper on the evaluation of computational creativity systems, Jordanous (2018) hints at a relation between quality and creativity:

> *These two concepts are highly interrelated, to the point that it is difficult (and perhaps inappropriate) to define creativity without incorporating quality judgements into that definition* [*of creativity*].

The author further mentions that many creative projects aim to produce high quality results, as can perhaps be expected. Bardzell, Rosner, and Bardzell (2012) related crafting quality in design to integrity, creativity, and public sensibility by conducting interviews with elite craft practitioners. Unfortunately, none of the craftsmen were software craftsmen, and creativity itself is only mentioned in passing. Within the field of computing education, we have not found any published work that investigates the potential correlation between the two concepts in context of student projects. We did find reports on the code quality of students (Börstler, Störrle, Toll, van Assema, Duran, Hooshangi, Jeuring, Keuning, Kleiner, and MacKellar, 2018; Keuning, Heeren, and Jeuring, 2017). However, creativity was not mentioned. The same is true for

reports on the creativity of computing students that do not mention code quality (Brookes, 2018; Salgian et al., 2013).

In order to get a better understanding of both concepts that might be related, we first provide an overview of what exactly clean code and creativity is, according to the current literature.

What is Clean Code?

Software, and therefore its source code, is in a constant state of change. New requirements result in pieces of code that are changed, refactored, and redesigned. Learning to program is one thing, but learning how to write code that is easily maintainable is something else. Martin (2009) offers in the first chapter of his book *Clean Code* several definitions of clean code from different well-known software experts. Many definitions are overlapping, of which the following concepts are the common divisor: (1) code that is easy to understand, (2) code that is easy to modify, and (3) code that is easy to test. Beck, Beedle, Van Bennekum, Cockburn, Cunningham, Fowler, Grenning, Highsmith, Hunt, Jeffries, et al. (2001) propose a fourth concept in *The Agile Manifesto*: Code that works correctly.

These concepts are even more important when working in teams on a shared code base. Since the clean code principles are prevalent in the software engineering industry, we argue that when introducing students to programming in CS1 courses, the necessary attention should be given to these simple but important rules. This can be done by gradually instructing students to write code in such a way that it is easy to understand and modify, as Vihavainen, Vikberg, Luukkainen, and Kurhila (2013) did in an apprenticeship-based CS1 course.

Keuning et al. (2017) say that so far, little attention has been paid to clean code issues in student programs. They found that novice programmers develop programs with a substantial amount of code quality issues, especially issues related to modularization. Keuning et al. use the term *"code quality"* to indicate the degree of cleanliness in students' source code. They cite Fowler's (1999) *Refactoring* book to provide an example of low quality in code, such as the code duplication "code smell". In our view, clean code, code

quality, and code smells are all closely related. In this paper, we consider a *clean coder* to be someone who is able to keep the code quality high by adhering to Martin's clean code principles—and thus also implicitly lowering the amount of Fowler's code smells. After all, code with a lot of duplication and other types of smell is not easy to understand, modify, or test.

13.2 Evaluating code quality & creativity

In this section, we elaborate our applied method used to assess both the creativity and the code quality of student projects. First, in order to help the reader interpret our results, we overview our CS1 course where the study took place.

The CS1 setting

The projects that were analyzed are part of a large-class CS1 course offered at our university (about 200 students yearly). The course has been designed for first-year engineering students. It is important to note that this course is part of a general engineering curriculum. Therefore, students who do not wish to specialize in computing, but like to pursue a chemical engineering degree instead, still have to pass the course. At our university, all courses of first-year engineering students are shared. This has two big ramifications. First, it implies that the course does not require prior knowledge of programming. Second, it means that the student group is very diverse, from students who would like to specialize in computing to students who prefer other engineering fields, such as chemistry.

The goal of the CS1 course is twofold. First, to introduce all engineering students to the basic concepts of object-oriented programming. Second, to lay the foundations for subsequent deeper learning of software engineering related courses offered to second-year students specializing in computing.

The course employs Java to explain the basics of program structures. During the first part of the course, BlueJ is used to introduce objects and relationships between them (Barnes, Kölling,

and Gosling, 2006). Typical topics of similar CS1 courses are covered, such as loops, variables, functions, iteration, but also classes and inheritance, as well as some less common ones such as exception handling, the very basics of threading, and the Model View Controller (MVC) pattern. During the second part, the NetBeans IDE is introduced, together with basic Graphical User Interface (GUI) programming using JavaFX and SceneBuilder. We find that by introducing something tangible such as JavaFX, students who do not enjoy programming or have no wish to further pursue computing still enjoy creating visible components. This allows the students to create little games as part of their project.

Four ECTS credits of the course are granted if students pass both their theoretical exam and their project. The project, which is an open programming assignment, accommodates for 33% of the total grade and thus is major part of the course. Students are free to form groups of at most three people. We encourage them to work together, but it is not mandatory to do so. The assignment itself is completely open. As mentioned in Chapter 12, open assignments are known to spark the creativity of students (Apiola et al., 2012) and to increase their motivation (Apiola et al., 2010). Students propose a subject themselves, which is then accepted or revised based on the complexity of their ideas.

This chapter analyzes the projects of students from academic years 2019–2020 ($n = 217$ students) and 2020–2021 ($n = 197$). In both years, the course was taught by the author of this dissertation. In total, 134 projects were processed, of which 110 were included in the study. The remaining 24 projects did not compile or bootstrap correctly, which made it impossible to judge the creativity, and therefore excluded from the study.

Evaluating code quality

To evaluate the code quality of the projects, we adapted the methodology from Keuning et al. (2017), who used PMD[1], a well-known static analysis tool that is able to detect a large set of bad coding practices in Java programs. PMD allows you to define *rules*, which are bad coding practices, that are detected and

[1] PMD is open source. The source and documentation is available at `https://pmd.github.io/`.

reported in a `CSV` format. In this paper, we define an *issue* as a problem detected by PMD in relation to a specific rule. Keuning and colleagues categorized their utilized PMD rules according to Stegeman, Barendsen, and Smetsers's (2016) developed rubric for assessing code quality. We used the same five categories to subdivide our own rules:

1. **Flow**; e.g. issues with nesting, paths and unreachable code.
2. **Idiom**; e.g. unsuitable choice of control structures, no reuse.
3. **Expressions**; e.g. overly complex expressions.
4. **Decomposition**; e.g. too large methods, duplicate code.
5. **Modularization**; e.g. too many methods, tight coupling.

PMD comes equipped with more than a hundred different rules. Therefore, we needed to select rules that were deemed relevant to both our study and our CS1 course. To aid with the decision, we randomly ran all 110 anonymized projects against both Keuning's PMD ruleset and the built-in "quickstart" PMD ruleset, which contains common best practice rules. First, all 58 rules were applied to the projects. Next, for each rule, we calculated the percentage of the total occurrence and the percentage of presence in the projects.

We discarded rules using the following criteria:

- A rule found in fewer than 5% of the projects (8).
- A rule found in fewer than 0.10% of the total errors (4).
- A rule related to documentation, presentation, imports or name conventions (e.g. `LocalVarNamingConventions`) (12).
- A rule marked as deprecated or controversial in the PMD documentation (e.g. `UnnecessaryConstructor`) (3).
- A rule deemed not relevant for our CS1 course (e.g. `UnnecessaryModifier` or `AvoidInstantiating ObjectsInLoops`) (5).

The final ruleset, containing 26 rules, categorized according to Stegeman's rubric, is visible in Table 13.1. All other rule settings have been left on their default settings, as per recommendation of Keuning et al. (2017).

For each project, PMD collected all issues regarding our selected 26 rules. Then, also for each project, we calculated the total amount

PMD rule/category	Description
Flow (6)	
CyclomaticComplexity	The number of decision points.
NPathComplexity	The number of acyclic exec paths.
PrematureDeclaration	
EmptyIfStmt	
EmptyCatchBlock	
ForLoopCanBeForeach	Prefer Java's foreach syntax.
Idiom (8)	
UnusedPrivateField	
SwitchStmtsShouldHaveDefault	
UnusedLocalVariable	
UselessParentheses	
CloseResource	Ensure closure of connections.
ControlStatementBraces	Enforce braces for control stmts.
UnnecessaryLocalBeforeReturn	Creation of unneeded variables.
UnusedFormalParameter	
Expressions (5)	
SimplifyBooleanExpressions	
CollapsibleIfStatements	Consolidate if statements.
SimplifyBooleanReturns	
ConfusingTernary	Negation problems in if/else clauses.
AvoidReassigningParameters	
Decomposition (3)	
SingularField	Fields with limited scopes.
NcssCount (Lines of Code)	Methods (60) and Classes (1500)
CPD-50	Copy-Paste Detector for 50 tokens
Modularization (4)	
LawOfDemeter	Low coupling: "Only talk to friends".
TooManyFields	
TooManyMethods	
GodClass	A class that does too many things.

Table 13.1: Selected PMD rules per category. Rules marked in italics also occur in the top 10 list of most frequently found issues, in Table 13.3.

of issues and the total amount of unique issues reported by PMD. For instance, if a project contains three unused private fields and two God classes, the total amount of issues is 5, while the amount of unique issues is 2 (`UnusedPrivateField` and `GodClass`). Irrelevant information, such as the the specific location of each issue, was discarded.

To cross-validate whether the selected PMD rules effectively measure the code quality, half of the projects from academic year 2019-2020 were randomly assessed manually by inspecting the anonymized source code. A score between 1 and 10 was assigned by the author of this thesis that indicates the *code quality rate* of these projects. The author has more than a decade of experience in

the software engineering industry and has guided many software development teams on how to write clean code. We found a strong negative correlation between the manual assessment and the amount of total PMD issues for each project ($r = -0.70, p = 0.00$) and a moderate negative correlation for the total number of unique issues ($r = -0.54, p = 0.00$). This signifies that the 26 PMD metrics reported in this chapter are solid ways to evaluate the global code quality of all student projects.

Evaluating creativity

As mentioned in Chapter 7, there is no uniform definition of creativity. Therefore, evaluating the creativity of student projects can be problematic. Rhodes (1961)' 4P Creativity Model defines four different dimensions that influence creativity: *Person*, *Press* (or *Place*), *Product*, and *Process*. Each dimension can be evaluated in its own way. For instance, to assess the creativity of a *person* (the student), one can rely on different personality tests. For this study, we are interested in the assessment of students' CS1 programming projects, which are tangible outcomes of their creative process, and thus are part of the *Product* dimension.

Our own CPPST from Chapter 11 measures creative achievement using self-assessment during the development of the project—it does not focus exclusively on the tangible outcome, even though some of the questions are related to it. In addition, we noticed that some first-year students interpret CPPST questions quite differently, making the reliance on self-assessment tests for this study an ill fit.

In the field of cognitive psychology, Amabile's (1982) CAT system is a common evaluation technique to assess the creativity of a product (Baer and McKool, 2009). This method relies on the opinions of expert judges that give a score between 1 and 10 for each product in a pool. Unlike many other techniques for creativity assessment, the CAT technique is not tied to any particular theory of creativity, and plays well with the socio-cultural dimensions of creativity. Since the CAT scoring process is highly subjective, to reconcile differences, it is recommended to form a panel of expert judges, and to calculate the average of the given scores (Baer and

McKool, 2009). Even though assembling a group of expert judges is very resource-insensitive, replacing all experts with novice judges, such as students, is generally not recommended (Kaufman, Gentile, and Baer, 2005).

For this study, we recruited both novice and expert judges. Two members of the CS1 teaching staff acted as expert judges, while five second-year students acted as junior judges. After explaining the design of this research and the CAT concept, each judge blindly evaluated the creativity of every anonymized project. After spending exactly one minute for each project, a score was administered individually, without looking at the source code. We refrained from providing a definition of creativity, as per recommendation by Baer and McKool (2009). The novice judges highly enjoyed evaluating projects of their fellow students. More than a few times, they made the remark *"I wish I spent more time working on my own project, now that I see all the others"*. Chapter 12 proves that engaging students in grading increases their motivation.

The Standard Deviation (SD) of the scores between the judges, averaging on 1.06, was low enough for us to conclude the same as Baer and McKool (2009): judges score surprisingly similar. Therefore, an inter-rater reliability is not relevant when using the CAT technique. However, as an extra verification step, when the judges reached a significant disagreement ($SD > 1.50$, 10 out of 110 projects), the project concerned was discussed in group, after which a new global score was assigned. The projects with the largest SD were suspected of plagiarism by a few judges, which resulted in a low score.

Finding relationships

According to a very recent systematic literature review on inferential statistics in computing education research, one of the most frequently used tests for inferring a relationship between ordinal and interval level data is the Pearson product-moment correlation (Sanders, Sheard, Becker, Eckerdal, and Hamouda, 2019). Since the creativity scores are ordinal (arranged on a scale from 1 to 10) and the PMD metrics can be considered as interval

level data (a continuous scale), we chose to answer our research questions by using this recommended correlation technique.

13.3 Results & discussion

We will first discuss creativity and code quality separately, before investigating potential relationships. The full data set of the 110 processed projects in both academic years can be consulted at `https://people.cs.kuleuven.be/~wouter.groeneveld/correl/`.

Projects and creativity

On average, the judges rated the creativity of the projects at 5.92 out of 10, with a global SD of 1.46. The normal distribution of the CAT scores is visible in Figure 13.1. The average is 13 out of 20, with a global SD of 3.37. There is a moderate correlation between the creativity score and the total grade of the project ($r = 0.46$, $p = 0.00$). This is to be expected, as creativity is one of the criteria in the evaluation rubric of the CS1 projects, next to others such as complexity and JavaFX UI design.

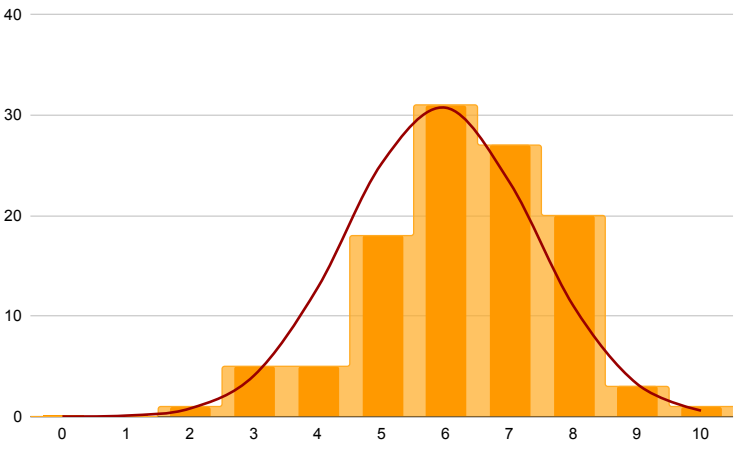


Figure 13.1: The normal distribution of the creativity CAT scores (x-axis) for all projects (y-axis).

We categorized each submitted project by topic. Since the assignment itself was open, it is interesting to see that many, but not all, students resorted to implementing popular small games. The topics, along their respective average scores, are visible in Table 13.2. Examples of simple boardgames we found were the Game of the Goose and Monopoly. Examples of simple dicegames we encountered were Yahtzee and higher/lower. We considered some projects to be more original than others, such as the Pokémon-inspired random battle games. However, since they were based on an existing game, and occurred more than once, we decided against categorizing them as "original". Furthermore, it is also interesting to note that original ideas do not necessarily mean high grades. For instance, students who played it safe and implemented a solid version of chess or checkers scored in total on average 15 out of 20, while completely original projects scored on average 12.18. This is because next to creativity, complexity is also a component of the project grade.

Topic	#	%	CAT	Grade
original	**20**	**18.18%**	**7.09**	**12.18**
doodle jump	16	14.55%	4.79	11.75
space shooter	15	13.64%	5.40	11.87
race/frogger	8	7.27%	6.51	13.94
arkanoid	7	6.36%	5.87	14.93
tic-tac-toe/4-in-a-row	7	6.36%	4.58	10.00
chess/checkers	6	5.45%	5.70	15.00
snake	6	5.45%	4.96	12.67
simple boardgame	6	5.45%	5.61	12.25
pacman	3	2.73%	5.46	13.67
bubble shooter	3	2.73%	6.57	16.67
simple dicegame	3	2.73%	4.36	9.67
poké-battle	2	1.82%	6.95	10.00
2048 game	2	1.82%	7.65	15.00
mastermind	2	1.82%	6.47	12.00
hang man	2	1.82%	6.14	9.50
tetris	2	1.82%	6.35	13.50
minesweeper	1	0.91%	4.67	14.00

Table 13.2: Identified project topics and their average CAT scores and total grades of the 110 projects.

Projects and their code quality issues

Table 13.3 contains a list of the top 10 most frequently found code quality issues by PMD. These issues are marked in italics in Table 13.1. We also counted the number of issues that frequently occurred in the same projects. First, the quotient (`Q`) of total issues in relation to unique issues found in each project was calculated. Next, for each project with `Q` higher than 10, issues that occur 15 times or more are counted, resulting in 11 projects. In total, 7363 issues have been found in 104,456 lines of code. The average amount of issues per project is 64, and each project contains on average 836 lines of code. We counted 10 unique issues on average per project. The low correlations in Table 13.3 hint at no relation between specific issues and a high CAT score.

Table 13.3: The most frequently found code quality issues, including project counts with a high number of recurring issues, and respective correlations with the average CAT creativity score. In total, 7363 issues were reported by PMD.

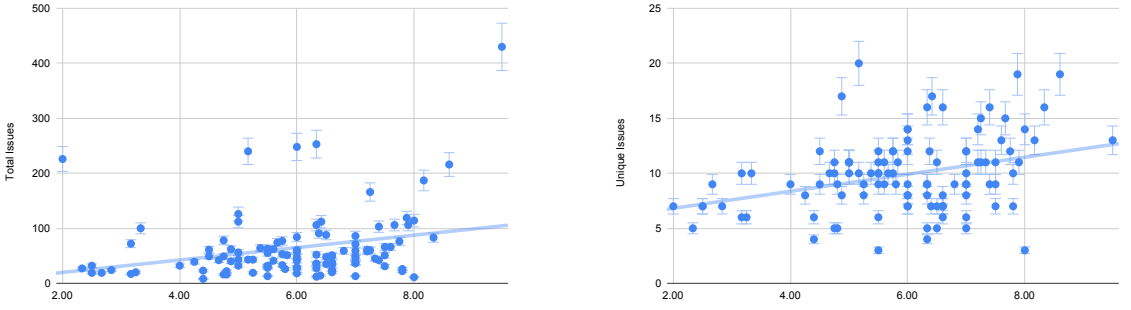
Issue	Category	#	Recur?	Corr.
LawOfDemeter	Modularization	4113	11	0.19
SimplifyBooleanExpressions	Expressions	646	4	0.26
UnusedPrivateField	Idiom	491	0	0.12
CPD-50	Decomposition	448	2	0.27
UselessParentheses	Idiom	411	0	0.09
SingularField	Decomposition	359	0	0.13
ControlStatementBraces	Idiom	327	2	-0.11
CyclomaticComplexity	Flow	242	1	0.36
SwitchStmtsShldHvDefault	Idiom	170	0	0.28
CollapsibleIfStatements	Expressions	156	2	0.02

When inspecting the top 10 PMD issues of the millions of student projects analyzed by Keuning et al. (2017), we see that `LawOfDemeter` also occurs often: it is the third most common issue. It is interesting to note that coupling-related issues are commonplace. In addition, `LawOfDemeter` is also the issue that is the most recurring within a single project.

RQ1—Relating creativity to code quality

We found almost moderate positive correlations between lines of code and evaluated creativity ($r = 0.38$, $p = 0.00$), between unique code quality issues and evaluated creativity ($r = 0.33$,

$p = 0.00$), and between total code quality issues and evaluated creativity ($r = 0.27, p = 0.00$). Although the correlations are not strong, they do seem to suggest that more creative projects contain more code quality issues. Figure 13.2 displays scatter plots of the CAT scores projected on the unique issues and total issues, including a linear trend line. If we remove the edge cases with high error bars from Figure 13.2, where the total issues of the projects exceeds 200, the correlation between the total code quality issues and evaluated creativity increases further from 0.27 to 0.34. The results of this study were informally shared among second-year students (who completed the CS1 project in 2019–2020) as part of an introduction into academic research. When asked for a possible explanation for the correlations, the students suggested two distinct possibilities.



(a) Compared to total issues found.

(b) Compared to unique issues found.

Figure 13.2: Scatter plots of evaluated creativity CAT scores compared to code quality issues.

First, it could be that students who decide to take on more original projects also experiment more. A consequence of experimentation during the development is encountering unforeseen bugs which students were not able to quickly resolve due to lack of experience. As a result, several code quality issues may have been overlooked while working as fast as possible to make up for the time lost looking for the bugs. This is in line with existing research that shows that first-year students are not yet competent enough to accurately plan and estimate software projects (Yuen, 2007). A lack of decent planning could be more damaging for creative projects, as our results indicate that the

higher the creativity, the higher the complexity. This is further elaborated in the next subsection.

Second, during the academic year, students have to complete multiple projects of multiple courses, not limited to computing. This rapid context switching potentially causes a heavy fragmentation of the students' time. Spending little fragments of time on a software project could be harmful to the code quality, compared to working on a piece of software without interruptions or distractions (Rodríguez, Partanen, Kuvaja, and Oivo, 2014).

The above reasons have merit but are only informal remarks of the students themselves, although supported by literature. We plan to continue the investigations of the rationale in future work.

RQ2—Are certain issues more relevant?

The top 10 found PMD issues listed in Table 13.3 reveals that some issues are more correlated to the CAT scores than others. Yet the possible interpretations of these numbers are dubious. For instance, why would `SwitchStmtsShouldHaveDefault` be more related to creativity than `LawOfDemeter`? Not every issue appears as often as others: students seem to make more mistakes against good coupling practices than to forget the `default` statement in a switch block, although we did not count the total amount of switch blocks in a project. We argue that some of the categorizes defined by Stegeman et al., as explained in Section 13.2, can have more severe consequences for Robert C. Martin's clean code concepts than others. Missing a `default` case would be less devastating for long-term code maintainability than violating the Law of Demeter.

Most issues show no moderate correlation with creativity, although `CyclomaticComplexity` ($r = 0.36$) nears 0.40. A cyclomatic complex project is a project where the code contains too many decision points in certain methods to be easily readable. In practice, this is usually visible by nesting multiple `if`, `for`, `while`, and `case` statements. There is an internal strong positive correlation between cyclomatic complexity and the issues `CPD-50` (Decomposition, $r = 0.67$), `LawOfDemeter` (Modularization, $r = 0.57$), and `NPathComplexity` (Flow, $r = 0.51$). Since the CS1 project assignments were open, we argue that some students

challenged themselves by picking a complex problem (e.g. a 2D platformer with gravity and enemies) instead of a simple one (e.g. a tic-tac-toe game). Their risk got rewarded by receiving higher creative scores from the judges. However, since these first-year students are still novice coders, a complex problem introduces many messy complexities in their code, hence the higher amount of aforementioned PMD issues, more duplication, and thus generally more lines of code.

13.4 Limitations

This study analyzes CS1 projects of two academic years. Having access to more data would potentially alter the normal distribution of Figure 13.1 or the correlations of Figure 13.2. We did not take student group compositions into account. Nonetheless, we think that the results of the 110 projects are significant enough to warrant the results described in Section 13.3. Another possible threat to validity could be the interpretation of the judges during the CAT phase. Our aim was to employ CAT to evaluate creativity and to employ PMD to evaluate quality. Jordanous (2018) warned about the potential confusion between creativity evaluation and quality judgments, leading to less grounded evaluative results. We believe that by cross-validating and by using multiple judges, as explained in Section 13.2, the effects of potential confusions have been mitigated.

An additional shortcoming could be focusing on the wrong PMD issues, thereby correlating creativity with irrelevant code quality issues. However, as we based our method on Keuning et al. (2017)'s work, and took great care in selecting the relevant PMD issues for our CS1 setting and based on the cross-validation, we are fairly certain that this eliminated the possible inclusion of irrelevant PMD issues. Another risk could be that certain excluded rules might be more related to creativity. Rejected rules, according to the criteria from Section 13.2, do not accurately represent the students' code quality, and therefore, were not deemed relevant.

13.5 Conclusion

We have analyzed 110 projects from two academic years to investigate whether there is a relation between creativity and clean code in CS1 student projects. Our results show preliminary evidence that the more creative projects are, the more code quality issues arise in these projects and the less clean the submitted code turns out to be. This potentially creates a new problem: the more we as educators encourage creativity, the more messy the produced code can be. Thus, it is vital that the necessary attention to the clean code principles is given. We argue that this should become more important as we let students experiment and be creative. This is in line with Keuning et al.'s (2017) conclusion.

If we want to increase the creativity of students' work, we need to pay attention to the quality as well. Future work might shed more light on the teaching of clean code in relation to creativity. For example, we do not yet know what the effects of explicitly teaching creativity are on the code quality. Most studies, like this chapter, limit the implementation of creativity in a CS1 course to the use of open assignments. Also, we wonder whether the observed effect would be reversed for last-year computing students, as they should be more disciplined in keeping their code quality high.

In the following chapter, we will explore how creativity can be improved by means of an an interdisciplinary intervention, where SE students have to work together with design students that are less technically proficient but perhaps more used to a holistic approach. The open assignment style of the project will be maintained, as will the CAT analysis of the projects. This time, our own CPPST assessment tool will be employed as well.

14 Interdisciplinary Creativity

Publication details

A shortened version of this chapter has been accepted in the *Proceedings of the 51th SEFI Annual Conference 2023* as *Does Interdisciplinary Creative Coding Boost Creativity? A Mixed Methods Approach* (Duyver et al., in press) and will be published shortly.
Its main contribution is the discovery of an increase in socio-interactive creativity levels when SE students have to work together with graphic design students on an open-ended creative coding assignment. Also, we observed fixed mindsets towards creativity ("design students are more creative than we") that call for future work to investigate and remediate.
In addition, a part of the results of this chapter were also published in the master's thesis of Duyver (2022).

While previous chapters explored increasing creativity by means of motivation and peer grading using open-ended assignments, in this chapter, we explore the influence of an interdisciplinary intervention on creative problem-solving skills. SE students and Graphic Design (GD) students have to work together on an open-ended creative coding assignment in the `p5.js` editor, an online JavaScript-based Processing Web Editor. 39

students were randomly paired and divided into three categories: SE+GD ($n = 18$) and the control groups SE+SE ($n = 10$) and GD+GD ($n = 11$). A mixed methods approach was taken to gather and interpret results: Amabile's Consensual Assessment Technique once again provided a global creativity score for the finished product, while the Creative Programming Problem Solving Test assessed three dimensions of the creative process[1], and 9 semi-structured follow-up interviews provided context and revealed underlying themes of the intervention.

[1] The factor analysis of Chapter 11 revealed the three dimensions as *Ability*, *Mindset*, and *Interaction*. See the respective chapter for more details.

In Chapter 8, we concluded that in order to foster creativity in higher education, three approaches can be taken:

1. *Introduce experimental learning*. Experiments with open-ended project-based learning have been shown to be beneficial towards students' creativity.
2. *Get students out of the classroom*. The environment in which the learning takes places plays a significant part in the creative potential of students.
3. *Put creativity first, programming second*. This opens up computing to a more diverse student population.

This chapter provides an experience report that combines all three approaches. Experimental open-ended learning is employed using creative coding and `p5.js`. We got students out of the classroom: the intervention took place outside of our own faculty, at the GD campus of the LUCA School of Arts in Ghent, 150 km away. And lastly, we put creativity first by refraining from grading and emphasizing on experimentation and having fun as a goal in itself.

We are interested in exploring ways to amplify creative problem-solving capabilities of SE students in higher education, beyond the conventional approaches. Since creativity can express itself in different ways, we ask the following questions:

Chapter Research Questions

1. *What is the influence of an interdisciplinary creative coding project on the creativity of the process?*
2. *What is its influence on the creativity of the end product?*

A mixed methods approach, as revealed in Section 14.2, was taken to explore these questions. Although organizing interdisciplinary projects is nothing new (Carter, 2014; Tchoua, Choi, Klasky, Liu, Logan, Moreland, Mu, Parashar, Podhorszki, Pugmire, et al., 2013), as far as we are aware, interdisciplinary coding assignments with emphasis on creativity using a mixed methods approach have not been conducted before.

First, we provide a brief overview of related work in Section 14.1, before explaining the approach and finally revealing and discussing the results in Section 14.3.

14.1 Background

This chapter touches upon three different themes: interdisciplinary collaboration, creative coding, and measuring creativity. We will briefly discuss the first two and direct the reader to Part III for a discussion on the definition and measurement of creativity.

Interdisciplinary collaboration

"Interdisciplinary computing classes are worth the effort", concludes Carter (2014). In that study, the interdisciplinary aspect aided in recruitment and retention of computing students, and provided opportunities to hone "soft skills". The paper mostly emphasizes collaboration and teamwork, but as revealed in Chapter 4, there are four groups of important soft or non-technical skills: communicative, collaborative, problem solving, and personal. While we also found collaboration to be a major component in our intervention, most existing interdisciplinary experience reports put too little emphasis on possible benefits of problem-solving skills.

Nissani (1997) published *Ten Cheers for Interdisciplinarity*, where creativity is one of the reasons to pursue interdisciplinary collaborations. We believe this to be especially relevant in the field of computing where the demand for interdisciplinary skill crossovers is growing, according to Carr, Jones, and Wei (2020),

who successfully intertwined computing with behavioral and social sciences:

> *Our approach of embedding computing education into the social sciences demonstrates a promising model of broadening participation in computing and meeting the nation's increasing demand for interdisciplinary computing workers in the digital age.*

However, again, there was no mention of creativity nor problem solving. The aforementioned systematic literature review on creativity in computing education only mentioned one paper that explicitly aims to amplify creative skills through interdisciplinary teaching. This was done by introducing creative writing (Jones et al., 2011). Yet the pilot project only used surveys to gather students' opinions, while our intentions are to measure both the creative process and outcome, next to conducting interviews after the intervention.

Creative coding

The concept "creative coding" is often employed in curricula to explore code as a medium for self-expression (Peppler and Kafai, 2005). The focus here is not sparking creativity to deal with daily programming problems, but rather to use code to express your creative urge. This is typically done using the Processing programming language and has been known to increase engagement and excitement for computing (Greenberg, Kumar, and Xu, 2012).

Processing is an excellent introduction to coding, as:

1. Its API is simple and very well-documented;
2. The JavaScript implementation, `p5.js`, has a web-based editor that negates the need to install compilers and tools;
3. The result can be easily shared using a simple link;
4. It is ideal for rapid prototyping;
5. It focuses on visual effects.

The focus on visual creations entices students, while at the same time offering a decent programming challenge.

14.2 Methodology

Before elaborating on the data gathering processes, to help the reader interpret the results, we first describe the courses and target groups involved in this study.

The setting

The intervention took place in the second semester of academic year 2021–2022. At our local faculty of Engineering Technology, second-year students who specialize in Electronics/ICT (a mix between hardware and software courses) or informatics (pure software) are required to follow the Software Engineering Skills (SES) course (6 ECTS credits), to which we applied the intervention. The whole group of 19 engineering students took part.

A neighboring university that showed interest in our project agreed to let their students, from the faculty of visual design, take part as well (20 students as pairs were needed to form). For them, this study was a part of their bigger second-year Digital Studio course (24 ECTS credits) that emphasizes experimentation, self study, and entrepreneurship.

Since none of the students had prior experience with `p5.js` or JavaScript, one of the co-authors organized an introduction session of two hours, two weeks before the intervention. By showing off some of the possibilities and building a small example together, we aimed to enthuse students and give them enough self confidence and pointers to ensure a quick start. In this study, the decision to use Processing was not made to explore creativity as a means for self-expression like in the work of Peppler and Kafai (2005), but as a means for problem solving.

To minimize unwanted side effects during measurements, all groups were placed in the same physical location, including control groups SE+SE and GD+GD. We chose the buildings of the design faculty as the location since we were most interested in the possible deltas of SE students and, as explained before, wanted to "get students out of the classroom". Of course, for design students,

the location didn't change. The final group composition can be found in Table 14.2.

As for the assignment itself, it was delineated, but not too much, as to leave room for creative freedom. Pairs had to create a visual exposition that emphasizes user interaction, for instance through sound, camera, or mouse input. The rest was left up to the imagination of the students. Examples shown in the introduction session helped to provide an idea on what is possible. The project was to be completed on a single day, that was kick-started by reiterating the goal of this study and pairing up students into random groups. One GD+GD group worked with three students.

Measuring the end result: CAT

Amabile's (1982) CAT was deployed to get a sense of the perceived creativity level of the SE+GD, SE+SE, and GD+GD groups. Well beyond the field of cognitive psychology, Amabile's CAT is commonly used to evaluate the creativity of an end product (Baer and McKool, 2009)–in our case, the `p5.js` exposition. CAT relies on expert judges that score the result between 1 and 10 in a pool. Unlike other creativity assessment tools, CAT is not tied to a particular theory of creativity, and thus encompasses all dimensions of CPPST as a more holistic approach. Since the scoring process is very subjective, it is recommended to recruit multiple judges and work with an average.

For this study, we enlisted seven judges that are part of the teaching staff of the involved courses: three computing experts (all co-authors), and four GD experts, of which two eventually opted out of the study. The judges were instructed in individually evaluating the creativity of every anonymized project by spending exactly one minute on each project. We refrained from providing a definition of creativity, as per recommendation by (Baer and McKool, 2009).

The standard deviation (SD) of the scores, on average 1.44, once again[2] mirrored Baer and McKool's (2009) conclusion: judges score surprisingly similar. Therefore, for CAT, inter-rater reliabilities are not relevant. However, as an extra verification step, when judges did not agree (threshold of $SD > 1.80$, 5 out of 19 or

[2] See the first thorough appliance of the CAT tool in Chapter 13, where a similar conclusion was reached.

26% of the projects), those projects were re-discussed in group, after which new scores were assigned and an SD calculated, until the threshold was reached.

Measuring The Creative Process: CPPST

Next to evaluating the end result with CAT, we were also interested in different aspects of the creative process. the CPPST tool allows us to assess whether or not our interdisciplinary intervention has effect on specific parts of students' creative problem-solving abilities.

The CPPST is a self-assessment test which was administered at the end of the project day. In it, students answer 56 questions on a Likert-5 scale, where a subset of the questions have been reverse-coded to prevent quick and mindless selections. The full question set and more information about CPPST is available in Chapter 11. A reduced set of 32 questions are enough to gauge the three factors of the CPPST, but we opted to include the full set as the extra data might help us in asking more specific questions in the semi-structured interviews.

The answers were collected using a Qualtrics web-based survey. To anonymize the results, we did not collect personal information except the type of group the student was part of. On average, the survey took 9.4 minutes, which is in line with the pilot study of Chapter 11 (9.5).

Enriching quantitative data: interviews

The different CAT and CPPST values are devoid of rich contextual information. Therefore, we decided to conduct additional interviews to put these numbers into context. The questions were composed after inspecting the results of the aforementioned tests.

After the CPPST survey results were collected, average and SD values were calculated for each question, grouped by the three student categories. Next, since we were most interested in the impact of the interdisciplinary component, deltas (δ) of averages between the subgroup SE from SE+GD vs SE+SE and the subgroup GD from SE+GD vs GD+GD were calculated. A $\delta > 0.5$ was marked as potentially interesting. For example, the question *"I*

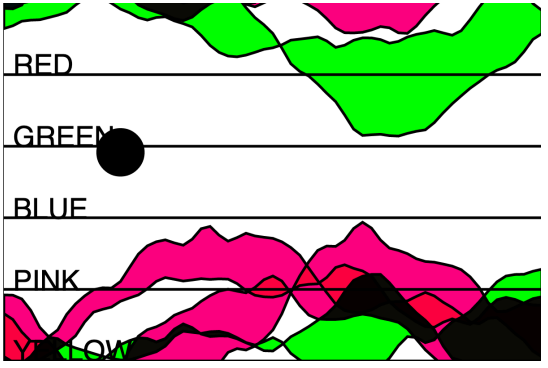
regularly asked feedback from my teachers" scored on average 2.33 by the SE+SE group, while the SE students from SE+GD scored 3.33 ($\delta = 1.0$). This could indicate that the SE+SE group was perhaps (too) confident. Big similar δ values like that call for qualitative context.

Hove and Anda's (2005) recommendations for conducting semi-structured interviews in empirical SE research combined with a discussion of the deviating δ values of the CPPST results guided us to the following question sets, divided into two distinct themes: **general context** (1–2) and more **in-depth**-related (3–9) questions. The questions, which include follow-up and clarification sub-questions, are displayed in Table 14.1.

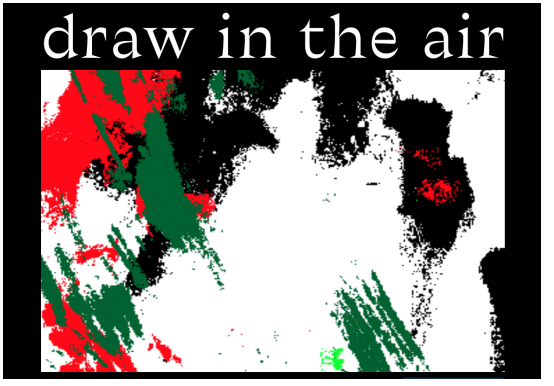
Table 14.1: The question set for the semi-structured interviews.

#	**General context** questions
1	What did you think of the experience?
...a	... What did you like the most?
...b	... What did you like the least?
...c	... How did your day go?
2	How did the collaboration go?
...a	... Compared to other group assignments?
...b	... Would it have been different if you were in another group?
#	**In-depth** questions
3	How did you tackle the project in general?
...a	... Would you have been more thorough if it was graded?
...b	... Did you divide the tasks in a certain way?
4	How did you tackle brainstorming and ideation?
...a	... How much time did you take to get to an idea?
...b	... Did you first draw something or immediately started coding?
5	What did you do when a problem occurred and you were stuck?
...a	... Did you give each other feedback?
6	What did you learn with this project?
...a	... At technical or design level?
7	In which way did you get out of your comfort zone?
8	How did you tackle the openness of the assignment?
9	What would you do different if you were to re-do it?

Our aim was to interview a random selection of 30% of the participants of each group. The interviews took place online, at most four weeks after the intervention. Data from the interviews was processed using qualitative coding as presented by Onwuegbuzie et al. (2009). First, all interviews were transcribed



(a) The "Color Waves" project.



(b) The "Draw in the Air" project.

Figure 14.1: Examples from two `p5.js` projects in progress.

and anonymized. Then, the transcripts were read multiple times independently by two co-authors to apply an open coding step, initially identifying 43 codes. Next, in order to identify patterns, notes were compared, cross-validated, and reduced into 25 codes grouped into 4 themes in an axial coding step. These themes served as a starting point for discussion and to cross-link back to the quantitative results.

14.3 Results & discussion

At the end of the project day, all pairs successfully created an interactive visual exposition using `p5.js`. To get a sense of the various creative projects built, we list a selection of the creations, together with two screenshots in Figure 14.1:

- Abstract art that react to sound or mouse input;
- "Draw in the air" using the camera;
- A text to music generator;
- 3D objects on a table that can be swiped away;
- A student that deliberately looks away from an approaching "DEADLINE" text controlled by the mouse;
- A multi-headed version of Snake;

All projects can be interacted with and are available at https://arneduyver.github.io/creative-coding/gallery. The

next subsections describe the results of the intervention, starting with an analysis of the quantitative data, followed by the qualitative data, ending with a holistic view that ties all results together.

Quantitative results

Table 14.2 summarizes average CAT and CPPST values for each group. When inspecting the overall CAT creativity grades, the test group (SE+GD; 6.30) performed less well than SE+SE (6.88), but better than the GD+GD group (6.04). Although the differences in grades can be considered small, it is still interesting to note that the interdisciplinary team scored less well than the pure SE team. Furthermore, the normal distributions of the CAT scores, as seen in Figure 14.2, reveal that the SE+GD grades were more varied compared to the very concentrated SE+SE group. This signifies consistent creative qualities of the end products for SE+SE, but does not provide any rationale. Since the CAT test is devoid of context and only yields a single number, we have to look at alternative creativity measurements to better understand these results.

Table 14.2: Group composition (number of students; groups) and average CAT and CPPST (Ability; Mindset; Interaction) creativity scores. In total, 39 students took part, divided into 19 groups.

Group	#	CAT	Ability	Mindset	Interaction
SE+GD	18;**9**	6.30	3.56	3.49	3.67
SE+SE	10;**5**	6.88	3.69	4.07	3.56
GD+GD	11;**5**	6.04	3.23	2.61	3.40

Since the CPPST measures three domains focused on creative problem solving, these averages (on a scale of 5) reveal more nuances than CAT does. This time, we see that the test group performed better at *Interaction* level (the lines in Figure 14.3 cross at the top right), but less well at the other two levels. In short, the socio-interactive components of creativity were stimulated more in the interdisciplinary group, while losing some *Ability* and quite a lot of *Mindset*. At the one hand, this sounds logical, since the SE+GD students do not know each other or each others' field of study. Yet remember that other groups were also assigned at random, and we assume SE+SE or GD+GD pairs should be able to

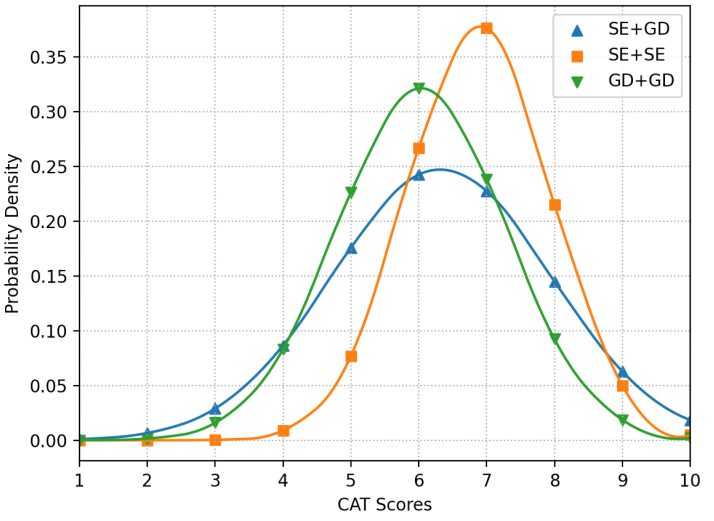


Figure 14.2: The normal distribution of CAT scores for groups SE+GD (△), SE+SE (□), GD+GD (▽).

more easily work together because of their shared education.

Another interesting result is the very low *Mindset* of the GD+GD students (2.61). It could be that they struggled to keep themselves motivated when encountering programming problems, since they had no access to a technically proficient SE student. It could also be that the coding assignment combined with a fixed mindset ("I can't code") resulted in an early defeat. At this point, we are still guessing, and in need of qualitative context, which will follow in the next subsection.

Qualitative results

In total, 9 students were interviewed: 5 from the SE+GD group (3 from SE and 2 from GD, thus 29.4% of that population), 3 from the SE+SE group (33.3%), and 1 from the GD+GD group (9%). Due to practical difficulties, only 3 GD students were interviewed, skewing the qualitative results in favor of SE students. Since we are mostly interested in the effects of the intervention on that group, and the computing education community is most likely to extract



Figure 14.3: The average CPPST domains for groups SE+GD (△), SE+SE (□), GD+GD (▽).

interesting findings from that part, we do not consider this a threat to the validity of the study. Table 14.3 represent paradata of the interviews.

Table 14.3: Collected interview paradata for each group. In total, 9 students were interviewed for 146 minutes yielding 22,809 words.

Group	#	Avg. Duration	Avg. Words
SE+GD	5 (29.4%)	16m	2619
SE+SE	3 (33.3%)	16m	2384
GD+GD	1 (9%)	17m	2561

Transcript analysis initially yielded 43 codes across all groups, reduced to 25 and categorized in 4 distinct themes: *Curiosity* (6), *Collaboration* (7), *Method* (7), and *Mindset* (5). We now briefly describe findings of each of the themes with codes emphasized in bold. We then end the discussion section with a summary to hark back to the CAT and CPPST results.

Curiosity

Almost all interviewees mentioned they liked the intervention because it allowed them to **explore beyond their education**. A GD

student said it was a "unique opportunity to learn how to program" and SE students mentioned they "had no idea it was that easy to create something visually". Especially **being in the same space** turned out to be inspirational, as students regularly got up to see others at work and mentioned ideas cross-pollinated quicker that way. The unique **physical environment**, as per recommendation in Chapter 8, indeed seemed to inspire SE students:

> *I was very impressed because as you entered, there were drawing tablets, 3D printers, a green room for photography, ...Then more ideas will come to you if you can look around to see what you can do.*

The student jokingly mentioned that our faculty should also buy that equipment, just to look at and get inspired by. Another trigger for curiosity was `p5.js` itself, as the **framework facilitated play**: students praised the clarity of the documentation and the easy to use web editor that facilitates experimentation through rapid feedback. Students liked fiddling with pieces of code to see what was possible. The **freedom** and absence of the stress in anticipation of a grade was also reported to play a role.

Interestingly, two GD students mentioned they got **frustrated** quickly and eventually zoned out, showing little interest in the assignment and sometimes even abandoning their partner. The drop in *Mindset* is clearly visible in Figure 14.3.

Collaboration[3]

SE+SE students working in a homogeneous group mentioned collaboration was **smooth**, even if they did not know their partner. Yet, for the heterogeneous SE+GD group, collaborating was **very difficult**. When asked why, several reasons were given: (1) It was "difficult to explain your own way of thinking to someone else"–this is a skill in itself that, compared to computing knowledge, receives little attention in our curriculum; (2) "I really had to drag it all out [of my partner]" could indicate a difference and possible incompatibility in personalities; (3) Some students wanted to get to know their partner, while others wanted to immediately start coding: "Working together with a complete stranger was hard".

[3] In Figure 4.3 of Chapter 4, we summarized expert panel opinions of non-technical skills, where *collaboration* was, next to *communication*, *problem solving*, and *personal* skills, a category that academics spent too little attention on. The study of this chapter could be placed in context of both that collaboration and problem solving category.

In the same vein, **giving feedback** proved to be a challenge as well, especially to other groups. Some SE+SE groups **pair programmed** and shared a laptop, while others started a competition with their peer groups. However, they did not give each other feedback. When SE+GD interviewees were asked whether they gave feedback to their partner, the answer from a GD student was:

> *We did show a bit to each other* [*on how we worked*]*, and the SE student was impressed by the 3D things* [...] *and I was looking a bit at how he did his coding, but I had the impression that we both did not really get what the other person was doing.*

The interviews confirm that the social aspects of co-creating was the factor that pushed students **out of their comfort zone**. Some homogeneous interviewees realized that collaborating—and possibly, the creative outcome—would be **very different in a mixed group**, but in a good way:

> *I found projects from the mixed group to be excelling in what the purpose was* [...] *they had ideas that were more interesting than ours.*

Although "more interesting" did not result in higher average CAT scores, all students do **acknowledge that collaboration is an important factor** to creative success.

Method

It was interesting to see a very **different approach** to tackling the assignment for SE+SE and GD+GD groups. The former look at examples, think about what they like to do based on the examples, and immediately start exploring that in code, while the latter first brainstorm for ideas, sometimes **iterating** over them, and only then look at what is possible to try and implement their ideas. This is a fundamental difference. SE students mentioned they had **difficulties with the openness of the assignment**, and with mixed groups, left the ideation part to their GD pair, as a SE student explained:

> *As to what we had to do, I asked her that, like what do you think we can do, because I had no clue, like okay we can do this or that. So I asked her. Like the direction we went for, I left that to her as well.*

For SE+GD groups, there was a **stereotypical task division**: "you do the code, I'll to the design", rarely daring to dip into the knowledge pool of their partner. Some SE+SE students (again **pairing**) admitted spending too much time trying to come up with a concept, struggling with what they wanted to create. A GD student from SE+GD vented her frustration in the interviews, as she wanted to ideate first, but their partner simply approved the first thing mentioned, thereby missing out on the critical and creative discussions.

As to what to do when stuck, SE+GD students mentioned they liked to:

> ***Get up and walk around*** [...] *I take that as a moment to think about something else* [...] *and usually come back with a solution.*

SE+SE students prefer **diving into the docs** before asking for help, while GD+GD students immediately enlist a teacher in case of coding troubles.

Mindset

Interviews revealed many **prejudices**: from "creating something visual is hard" (SE) to "I can't code" (GD). As much as everyone acknowledged **importance of heterogeneity in teams** and lauded the experiment as **refreshing**, we discovered a **fixed mindset** when it comes to creativity. SE students say "design students go more in the creative direction while we use logic steps to solve things", while GD students say "coding is very mathematical" and imply it is less creative than their visual work. Clearly, this fixed mindset can be very damaging to the creative potential of SE students. Also, students have a **wrong image** of what an engineer or designer does, as a GD student testified:

> *I think we're more the people who ask questions while you just tell an engineer to do this or that and he'll understand it that way.*

Summary

On average, the CAT jury graded SE+GD less well than SE+SE. The CPPST test showed that they indeed performed a little worse

on *Ability*, a lot worse on *Mindset*, but better at *Interaction*. The interviews revealed that the coding hurdle was very high for GD students. This, together with the fixed mindset attitude, explains the drop in *Mindset*. Yet, although the SE+GD group introduced plenty of collaboration issues, it also allowed for more ideation and forced SE students to explain something technical in a non-technical way, as a SE student testifies:

> *I think the biggest thing I have learned from this is that maybe later on it will also be a bit of the same style that when you go to customers, they don't really know what is realistic and what isn't, and that I may have already had a taste of that.*

14.4 Limitations

This chapter reports on an intervention with a relatively small group ($n = 39$) that took place within one academic year. It could be that larger groups or repeated interventions alter the averages mentioned in Section 14.3, although judging from the recurring themes throughout the interviews, we believe that the findings of this chapter will persist. It is also important to note that although the average CAT of SE+SE is quite high, it is likely to drop when the experiment would have taken place in our university, as students would not be exposed to a different environment, possibly sparking less inspiration, as also mentioned in the interviews.

Lastly, since the collaboration between different student groups caused friction, a prolonged intervention of for instance a week could iron out the initial acquaintance difficulties. We suspect that the CPPST *Interaction* δ between the groups would even be bigger then, possibly also increasing *Ability*, but the creativity prejudices would likely remain. Future work might shed more light on this.

14.5 Conclusion

This study explored the influence of an interdisciplinary intervention on creative problem-solving skills by pairing up SE

students with GD students. A mixed methods approach helped in identifying and understanding the various effects of the intervention. While we observed a slight decrease in the creativity of the end product (RQ1), the CPPST reveals that although *Mindset* needs more work, our intervention effectively increased the *Interaction* part of the creative problem solving process (RQ2).

It is important to note that due to the various reasons touched upon while discussing the qualitative results in Section 14.3, creativity can initially take a hit during interventions. This should not worry educators, but is something to be aware of. Using a measurement such as CAT is not enough to reveal the underlying constructs behind the numbers. As many students testified, the intervention was a great experience to get a taste of *real* creative collaboration, even though this is not visible by looking at the CAT results. And yet, the SLR of Chapter 8 revealed that most computing education studies on creativity employ a single metric.

We also found the fixed mindset approach towards creativity to be very problematic, especially for the teaching staff who try to improve the creative skills of students. This has also been noted by Apiola and Sutinen (2020) and ourselves in Chapter 11. In our view, this requires immediate attention from the computing education community.

The research questions posed in the chapters of Part IV also provide an answer to overarching RQ3 of this dissertation, which was:

PhD Research Question 3

How can we improve upon education to reduce the skill gap?

In Chapter 12, we saw that the motivation, curiosity, and constraint-based thinking skills of SE students can be improved by introducing limited hardware such as the GBA and involving students in the evaluation system.

Chapter 13 warned about the merits of introducing creativity as part of open assignments in CS1 courses. As first-year students' creative scores increased, so did the reported code quality problems, meaning we cannot simply up the creativity volume and call it a day.

Finally, in this chapter, we saw that the communicative and collaborative aspects of creative problem solving can be improved by means of interdisciplinary projects, even though the mostly fixed mindset of many students limited its effects.

A Conclusion

YOU HAVE MADE IT to the conclusion—thank you for your patience and interest in this work!

We started our adventure in Chapter 1 with the vague IEEE Standard Glossary of Software Engineering Terminology that defines software engineering as "the systematic appliance of engineering methods to software development" (IEEE Computer Society, 1990). During the course of the exploration of these engineering methods, we have gradually discovered that it takes more than technicality to succeed as a software engineer.

In fact, it takes a bunch of non-technical skills to do so. In Part I, we explored what exactly those non-technical skills are. Once we identified a list, in Part II we wondered: how big is the requirements gap between on the one hand expectations of SE professionals and on other hand the current state of SE higher education? After discovering creativity as a crucial but seemingly missing skill, Part III pinpointed its meaning and context, while in Part IV we explored possibilities for improving education by various means of introducing creative problem-solving skills.

We will briefly reiterate the central research questions posed in this dissertation before ending with possible future work and the introduction of the "little sister" of this dissertation, specifically geared towards professionals.

A.1 What are non-technical requirements?

Overarching research question 1 (RQ1) was:

PhD Research Question 1

What exactly are the non-technical industrial requirements of modern software engineers?

In Chapter 2, we summarized a selection of similar terms, ranging from *soft skills*, *21st century skills*, *professional competencies*, to *non-cognitive abilities*. We ended up sticking to the term *non-technical skills*. First, the term "skills" is preferable to knowledge or competencies, as knowledge usually signifies technicality and competencies is too broad according to the OECD (2018) framework. Second, the term "non-technical" was preferred by our interviewees, as *soft skills* usually emphasize communicative skills and *professional skills* are too broad, including technical skills.

In Chapter 3, we reviewed which non-technical skills are identified by educators as important to teach software engineering students, and how have those skills been successfully taught. The SLR, based on 26 papers, resulted in a list of non-technical skills extracted from published teaching success stories, such as *conflict resolution*, *creativity*, *role awareness*, *empathy*, and *teamwork*.

In Chapter 4, we wondered which non-technical skills exactly are needed for a developer to excel in the SE industry, what the relative importance of these skills could be, and whether or not the opinion of industry experts differ from academics. This again resulted in a list of non-technical skills, this time extracted from the opinion of 36 professionals using a Delphi method, grouped into four skill categories: *communication*, *collaboration*, *problem-solving*, and *personal* skills. *Creativity* ranked third in two out of three panels, next to *curiosity* and *systematically verifying assumptions*[1].

[1] We consider this to be a part of creativity. A whole chapter of *The Creative Programmer* (Groeneveld, 2023) is devoted to critical thinking.

A.2 How big is the industry-academia gap?

Overarching research question 2 (RQ2) was:

PhD Research Question 2

How big is the gap between those requirements and software engineering education?

In Chapter 5, by manually curating 278 computing syllabi from 110 universities in 30 European countries, we explored which non-technical skills are currently taught in undergraduate computing courses, according to the learning outcomes, and what types of courses facilitate these skills the most. *Creativity* occurs less than 5% in the processed course syllabi, while the expected *teamwork* and *communication* skills appear more than 25%.

In Chapter 6, the data of all previous chapters was collated into one large gap analysis study, resulting in a top 3 list of biggest non-technical skill gaps in the field of SE: *continuous learning*, *creativity*, and *solution-oriented thinking*. Follow-up interviews with eight industry professionals revealed additional contextual insights, grouped into six overarching themes: *stimulating continuous learning*, *stimulating creativity*, *creative techniques*, *addressing the gap in education*, *skill requirements in industry*, and the *industry selection process*.

We have chosen *creativity* as the skill to focus on for the last research question, as our second SLR indicates it is severely underrepresented in computing education research.

A.3 How to improve education to reduce that gap?

Overarching research question 3 (RQ3) was:

PhD Research Question 3

How can we improve upon education to reduce the skill gap?

Before being able to answer RQ3, in Part III, we first explored the context and boundaries of creativity in SE.

In Chapter 7, we provided an overview of past and current research trends of the term *creativity* within and outside the domain of cognitive psychology, where its many definitions evolved from divergent thinking, originality and quality, to a socio-cultural verdict.

In Chapter 8, a second SLR explored the landscape of research on explicitly teaching and assessing creativity context of computing in higher education. After analyzing 301 papers, 58 of

them were categorized as *passing*, *peripheral*, or *in-depth* creativity-related studies. The SLR yields surprisingly little results, hinting at the scarceness of interdisciplinary creativity research published in computing venues. The chapter ends with a summary of three approaches taken to foster student creativity.

In Chapter 9, we explored what exactly the role of creativity is in the world of SE, and what makes one software developer very creative and the other less so. The focus group study with 33 industry experts summarized creative problem solving into seven domains: *technical knowledge*, *communication*, *constraints*, *critical thinking*, *curiosity*, *creative state of mind*, and *creative techniques*.

In Chapter 10, the questions from the focus groups were also posed to 10 SE students across multiple Flemish universities using in-depth interviews, after which the similarities and differences in perception of creativity between both worlds was analyzed by reinterpreting and comparing the data from the previous chapter. Interestingly, when it comes to interpreting creativity in context of programming, students put different emphasis on themes such as *when to be creative*, *why should you be creative*, *what is creativity like*, *how to be creative*, and *where does it manifest*.

In Chapter 11, the CPPST tool is presented, a self-assessment tool is for computing students in higher education to measure their creative programming skills. The tool encompasses the seven dimensions of creative problem solving as discovered in the focus group study, which a principal axis factor analysis groups into three overarching constructs: *ability*, *mindset*, and *interaction*.

The last part of this work provides suggestions to tackle RQ3.

In Chapter 12, as motivation is needed to be creative, we explored the effects of peer assessment on self-evaluation, motivation, and study time. In addition, the amplification of constraint-based thinking is explored with the help of Nintendo's GBA. We discovered that peer evaluation helps students understand and evaluate their own project, boosts their motivation to work on it, and increase the relation between motivation and time spent on the project.

In Chapter 13, we discovered preliminary evidence for a negative relationship between the perceived creativity and the code quality

of 110 first-year student projects, thereby calling for the necessary attention to clean code principles as creativity is promoted in higher education.

In Chapter 14, we explored the influence of an interdisciplinary creative coding project on the creativity of the process and end product using a mixed methods approach. While we observed a slight decrease in the creativity of the end product, the CPPST reveals that although *Mindset* needs more work, our intervention effectively increased the *Interaction* part of the creative problem-solving process. Additionally, SE students showcased a fixed mindset towards creativity but a growth mindset towards coding. In our view, this deserves immediate action from the computing education community.

A.4 Creative programming in practice

THERE IS NO DENYING the theoretical nature of this work, of which the interest will be most likely limited to the academic computing education research community. That would be a big shame: not just because of all the work put in, but also because of the missed opportunity to effectively increase various creative (programming) problem-solving skills as presented here.

That is why ***The Creative Programmer*** (Groeneveld, 2023) was conceived as a practical handbook specifically written for SE professionals of any experience level[2]. Each chapter corresponds to one of the seven dimensions of creativity. *The Creative Programmer* is a blend between a scientific understanding of creativity and how it translates to programming and actionable advice and thinking exercises that will make you a better programmer.

[2] The handbook also serves as valorization and knowledge transfer for this research.

From the product page of the publisher:

> *In The Creative Programmer, you'll learn the processes and habits of successful creative individuals and discover how you can build creativity into your programming practice. This fascinating new book introduces the seven domains of creative problem solving and teaches practical techniques that apply those principles to software development. Hand-drawn illustrations, reflective thought experiments, and brain-tickling example problems help you get your creative juices flowing. You'll soon be thinking*

> *up new and novel ways to tackle the big challenges of your projects.*
>
> *Each chapter in The Creative Programmer introduces you to a new theme of creativity that's derived from scientifically sound research. Discover the importance of communication, how constraints can make you more creative, methods to improve your critical thinking, and more. Short stories, examples, and exercises help you understand each new idea and clearly demonstrate how you can apply them to programming. You'll even be able to track your progress against a scientifically validated Creative Programming Problem Solving Test! Along the way, you'll enjoy examples and stories that show what makes creative technical geniuses tick.*

`https://www.manning.com/books/the-creative-programmer` contains more information including a sneak peak at the first few chapters. Grab you copy today! Promotial code `au35gro` adds a discount of 35%. You are welcome to reach out and send feedback to help improve possible future versions of the book.

The Creative Programmer is an excellent conversation starter in reading groups, adds leverage when convincing your manager to make room for more creative freedom, and helps both (under)graduate SE students and professional developers better understand the various theoretical and practical implications of creativity.



THE Creative Programmer

Wouter Groeneveld

MANNING

A.5 Suggestions for curriculum design

The teaching staff could help students realize their full creative potential by further implementing our findings in academia. In this section, we present a few suggestions to get started.

First, there is the question of whether or not to formally include creativity as one of the evaluation criteria. If we don't, since many students are keen to learn only what is the absolute requirement for a course as presented in the ECTS fiches, we risk neglecting the importance of creativity. If we do, we risk rewarding only students that already showcase creativity, for instance by evaluating the result using the CAT test. Our own CPPST test does not help here as it was not designed as a formal evaluation tool and is rendered useless when reduced to a single digit.

Instead of focusing on classical evaluation models, the teaching staff could instead add a required module that makes the student reflect on their creative endeavor throughout their studies. The CPPST test results vary from project to project as it focuses on the current process, making it an excellent tool for self-reflection. When it comes to evaluation, the result, a high or low score on one of the seven creativity domains[3], matters not. This could be coupled with an obligatory report where the student reflects on the results of the test: why was the score for *collaboration* low but for *technical knowledge* high? Why do these outcomes change when filling in the test in context of another course? Why do I (not) view myself as a creative individual?

[3] The factor analysis of Chapter 11 proved that the seven creativity domains of Chapter 9 could be compressed to just three domains: ability, mindset, interaction. Yet for an elaborate self-reflective exercise, it would be more interesting to stick to the seven domains of the initial CPPST test. See the spider chart of Figure A.1.

Again, for the CPPST, it is of little use to try and identify a growth curve from undergraduate to graduate. Instead, we as educators should focus on helping students break free of their mostly fixed creative mindset and help them understand that creativity is, just like programming, an attainable skill. Self-reflection combined with motivational creative assignments like the Game Boy Advance ones from Chapter 12 could go a long way, provided the exercises are not anxiously shielded greenfield projects.

Of course, for the reflective exercise to make any impact, students have to get to know the systemic aspects of creativity. If they are

presented with a spider graph at the end of a CPPST test, such as in Figure A.1, they should be able to interpret the results. The survey result webpage does summarize actionable advice for each domain as to what it means to have a low or high value, but it purposely stays vague, as each person and project is unique.

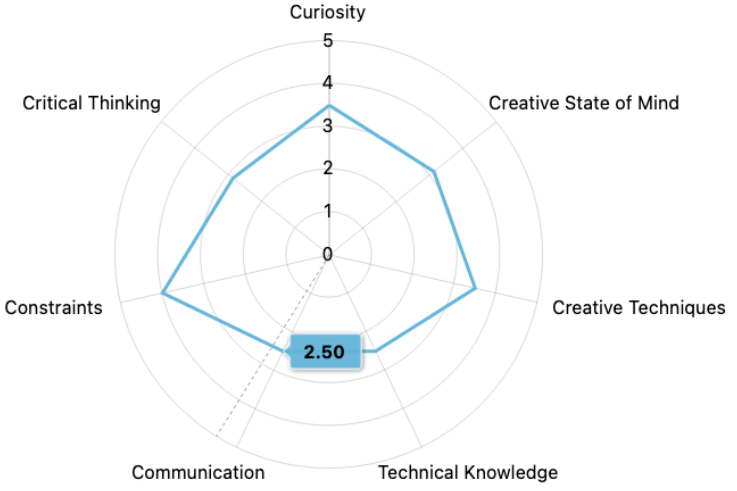


Figure A.1: A spider graph visualizing the CPPST results after filling in the survey at https://brainbaking.com/cppst/.

Therefore, at least a very brief exposure to the concepts as revealed in this dissertation is required. This does not have to be a sequence of boring lectures as part of a separate course called "creativity for programmers"—it could just as well be integrated in an existing course, or even better, regularly touched upon in multiple courses throughout multiple academic years. As to the material to use, the handbook *The Creative Programmer*, introduced in the previous section, is a superb way to start. While it was initially written with industry professionals in mind, it is also a great fit for students, as:

- It summarizes the research presented in this work in a digestible and succulent way[4] in roughly 200 pages;
- It bridges the research with practice using ample examples from the real world, including ones from the game development

[4] The fact that the book was critically reviewed and edited by non-academics (the development editor, the copy editor, the proofreader, ...) can only be called an advantage here.

industry, that is sure to spike the motivation of students;
- It contains multiple exercises that challenge your view of creativity and current problem-solving practices that can be easily translated to the classroom;
- It is a great way to jump-start discussions in class using the text as a shared base level;
- It helps students regularly hark back to the principles of creativity as these stay relevant, well beyond the safe boundaries of a single course, semester, or academic year.

Many curriculum implementation reports with emphasis on "21st Century Standards", as mentioned by Alismail and McGuire (2015), push *problem-based* learning forward as the solution to the skill gap problem. While that is a good start indeed, a much more interesting approach with regard to creativity would be *problem-defining*. Instead of handing the problem and having students use their divergent thinking skills to come up with original solutions, perhaps we should hand them a brownfield project and let them use their creative thinking skills to first identify the problem.

Other ways to spark the creativity of students would be to teach with emphasis on concepts using a few frameworks, programming languages, and technologies, and have them work with completely different frameworks, languages, and technologies. That way, cross-domain knowledge transference and critical thinking in coping with the unknown becomes an essential part of the assignment. It also much better reflects the ever changing real development world. Tate's (2010) *Seven Languages in Seven Weeks* is the perfect example of this. Teach students the concepts of software engineering using a programming language like Java, something they are familiar with in from their CS1 course, and design assignments in Go, Rust, Elixir, Kotlin, or Lua.

A possible threat preventing successful implementation in the academic curriculum is perhaps the teaching staff themselves. For example, a *Seven Languages in Seven Weeks* model requires both willingness and effort to diverge from classic assignments that are easier to grade and perhaps a bit too safe and familiar. Furthermore, letting creative problem-solving seep in every

possible nook and cranny of the curriculum involves close collaboration with other staff members—something that academia is not always particularly good at (Miller, 2014b). Yet before being able to help awake students' inner creative self, we as teachers first have to learn and express creativity ourselves.

To summarize, curriculum designers could use the CPPST test in combination with an obligatory self-reflection report to force students to stop and think about the various systemic parts of creative problem solving. To maximize the potential outcome, the many practical examples and exercises in the handbook *The Creative Programmer* could be used in introductory lectures on creativity in software engineering to introduce and situate the seven dimensions of creativity. Lastly, programming assignments should also be designed in such a way that students are encouraged to work with the mechanics of the creativity domains. And perhaps we as teachers should not ignore the fact that we cannot teach creativity without expressing it ourselves.

A.6 Suggestions for future research

As any self-conscious academic publication by convention seems to end:

More research is needed.

Reflecting upon possible future research work, we list a selection of the possibilities:

- Similar interdisciplinary or otherwise educational interventions could be repeated at a larger scale during several days;
- The usage of the CPPST tool at a larger scale or at companies with a second iteration of the factor analysis;
- Emphasize on helping evolve students' fixed mindset of creativity into a growth mindset;
- Longitudinal studies following first-year students throughout the SE programme might shed more light on how they perceive creativity;

- Interviews with the teaching staff can supplement the professionals' and students' perception of creativity;
- A part of our results could be translated for college students and generalized into computational creativity;
- The syllabi analysis study could be repeated over time to help discovering the evolution of non-technical skills in learning outcomes;
- More interdisciplinary research collaborations could help clarify the domain-generality of creative problem-solving skills—for instance, how do chemical/mechanical engineering or even architecture students perceive creativity?;
- Further disseminating the results and practices presented in this work and in *The Creative Programmer* across organizations, universities, and countries could be hastened with the involvement of accreditation boards;
- Exploring the evolution and influence of students' personality and affects in relation to creativity;
- ...

If you have an idea on how to help—or even better, want to collaborate—please reach out!

List of Figures

1 An early sketch of the 2018 PhD proposal. 6

1.1 Positioning the Engineering Technology programme. . . 21

2.1 The Dreyfus model of skill acquisition. 28
2.2 The OECD education 2030 framework. 31

3.1 A heatmap of different skills, aspects and levels. 48
3.2 A chord diagram linking creativity and teaching forms. . 51

4.1 The Delphi survey structure. 57
4.2 The ranked skills from the Delphi study. 62
4.3 Comparing opinions about skills between expert panels. 66

5.1 A color-coded map of university course categories. . . . 79

6.1 The context of previous and current chapters. 91
6.2 The skill mapping procedure explained. 92
6.3 The full skill comparison chart. 97
6.4 The skill comparison chart highlighting deltas. 98
6.5 A mind map that summarizes the themes we identified. . 100

7.1 An illustration of a TTCT test. 118
7.2 Amabile's Componential Model of Creativity. 120
7.3 Gero's creativity design spaces. 122

8.1 Scatter plot of papers included in the creativity SLR. . . . 132

9.1 The focus group data processing steps. 148

9.2 Focus group theme distribution rates. 149
9.3 The creative problem solving mind map. 150

10.1 The context of previous and current works. 166
10.2 Indications of thematic data saturation. 168
10.3 Perceptions of creativity summarized in a mind map. . . 176

11.1 Scree plot of the CPPST. 198

12.1 The GBA course poster as presented at SIGCSE 2020. . . 211
12.2 Scatter plots of self-evluation vs teacher-evaluation. . . . 217
12.3 A selection of the created GBA games. 218
12.4 A time graph of students' invested project hours. 221

13.1 The normal distribution of code quality CAT scores. . . . 234
13.2 Scatter plots of CAT scores vs code quality. 237

14.1 Examples from two `p5.js` projects in progress. 249
14.2 A normal distribution of interdisciplinary CAT scores. . . 251
14.3 The average scores of the three CPPST domains. 252

A.1 A CPPST result spider graph. 267

List of Tables

3.1 Identified skills to teach students. 42
3.2 Levels and teaching aspects from the SLR. 45

4.1 Panel group compositions and response rates. 56
4.2 The 11 identified communicative skills. 60
4.3 The 16 identified collaborative skills. 60
4.4 The 13 identified problem sovling skills. 61
4.5 The 15 identified personal skills. 61

5.1 A summary of included results in this study. 80
5.2 Non-technical skills identified in the SLR. 82
5.3 A categorization of encountered CS programmes. 83

6.1 The top three skill gaps for each study. 98
6.2 Collected interview paradata. 99

8.1 Paper classifications per category. 130
8.2 *Depth* papers: major themes 134
8.3 *Depth* papers: creativity measurement instruments 137

9.1 Collected focus group paradata. 149
9.2 The identified themes and subthemes on creativity. 151

10.1 Collected interview paradata. 169
10.2 Emerged creativity subthemes after coding steps. 170

11.1 Rotated Component Matrix for the CPPST. 198
11.2 Reliability item statistics (1/2). 199

11.3 Reliability item statistics (2/2). 200

12.1 The qualitative evaluation rubric. 214
12.2 Comparison of self and teacher assessment. 216
12.3 Self-reported means for ease of grading and accuracy. . . 216
12.4 Motivators and demotivators for the GBA project. 220
12.5 Correlation matrices for year 2018–19 and 2019–2020. . . 221

13.1 Selected PMD rules per category. 231
13.2 Identified student project topics. 235
13.3 Top encountered code quality issues. 236

14.1 The question set for the semi-structured interviews. . . . 248
14.2 A breakdown of student group compositions. 250
14.3 Collected interview paradata for each group. 252

List of Acronyms

ABET Accreditation Board for Engineering and Technology

CAT Consensual Assessment Technique

CFA Confirmatory Factor Analysis

CFI Comparative Fit Index

CPAC Cognitive Processes Associated with Creativity

CPPST Creative Programming Problem Solving Test

CPS Creative Problem Solving

CS Computer Science

CS1 Computer Science 1

CSE Computer Science Education

CSI Creativity Support Index

CTS Computational Thinking Scale

ECTS European Credit Transfer and Accumulation System

EFA Exploratory Factor Analysis

ESCO European Skills Competences qualifications and Occupations

ET Engineering Technology

FGS Focus Group Study

GBA Game Boy Advance

GD Graphic Design

GUI Graphical User Interface

ICER International Computing Education Research

LO Learning Outcomes

MVC Model View Controller

OECD Organisation for Economic Co-operation and Development

RMSEA Root Mean Square Error of Approximation

RQ Research Question

SD Standard Deviation

SE Software Engineering

SES Software Engineering Skills

SIGCSE Special Interest Group in Computer Science Education

SLR Systematic Literature Review

SWEBOK Software Engineering Body Of Knowledge

SWEBOS Software Engineering Body Of Skills

TOCE Transactions On Computing Education

TTCT Torrance Test of Creative Thinking

Bibliography

Cheryl L Aasheim, Susan Williams, and E Sonny Butler. Knowledge and skill requirements for it graduates. *Journal of Computer Information Systems*, 49(3):48–53, 2009.

Cheryl L Aasheim, Jordan Shropshire, Lixin Li, and Christopher Kadlec. Knowledge and skill requirements for entry-level it workers: A longitudinal study. *Journal of Information Systems Education*, 23(2):193, 2012.

Iqra Abdullah, OMAR Rozeyta, and Siti Aisyha Panatik. A literature review on personality, creativity and innovative behavior. *International Review of Management and Marketing*, 6(1):177–182, 2016.

ABET. Criteria for accrediting computing programs, 2019 - 2020, November 2018. URL https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2019-2020/.

Lingma Acheson and Ryan Rybarczyk. Integrating career development into computer science undergraduate curriculum. In *Computer Science & Education (ICCSE), 2016 11th International Conference on*, pages 177–181. IEEE, 2016.

Halah Ahmed Alismail and Patrick McGuire. 21st century standards and curriculum: Current research and practice. *Journal of Education and Practice*, 6(6):150–154, 2015.

Teresa M Amabile. Social psychology of creativity: A consensual assessment technique. *Journal of personality and social psychology*, 43 (5):997, 1982.

Teresa M Amabile. The social psychology of creativity: A componential conceptualization. *Journal of personality and social psychology*, 45(2): 357, 1983.

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

Susan Amato-Henderson, Amber Kemppainen, and Gretchen Hein. Assessing creativity in engineering students. In *2011 Frontiers in Education Conference (FIE)*, pages T4F–1. IEEE, 2011.

Aamir Amin, Shuib Basri, Mohd Fadzil Hassan, and Mobashar Rehman. A snapshot of 26 years of research on creativity in software engineering-a systematic literature review. In *International Conference on Mobile and Wireless Technology*, pages 430–438. Springer, 2017.

Christina Andersson and Doina Logofatu. Using cultural heterogeneity to improve soft skills in engineering and computer science education. In *Global Engineering Education Conference (EDUCON), 2018 IEEE*, pages 191–195. IEEE, 2018.

Marc Andreessen. Why software is eating the world. *The Wall Street Journal*, 20(2011):C2, 2011.

Mikko Apiola and Erkki Sutinen. Mindset and study performance: New scales and research directions. In *Koli Calling'20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, pages 1–9, 2020.

Mikko Apiola, Matti Lattu, and Tomi A Pasanen. Creativity and intrinsic motivation in computer science education: Experimenting with robots. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '10, page 199–203, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605588209. 10.1145/1822090.1822147. URL https://doi.org/10.1145/1822090.1822147.

Mikko Apiola, Matti Lattu, and Tomi A Pasanen. Creativity-supporting learning environment—csle. *ACM Transactions on Computing Education (TOCE)*, 12(3):1–25, 2012.

Mark Ardis, Shawn Bohner, Dick Fairley, Massood Towhidnejad, and Art Pyster. Graduate software engineering 2009 (gswe2009) curriculum guidelines for graduate degree programs in software engineering. *Integrated Software and Systems Engineering Curriculum (iSSEc) series*, 2009.

Mark Ardis, David Budgen, Gregory W Hislop, Jeff Offutt, Mark Sebern, and Willem Visser. Se 2014: Curriculum guidelines for undergraduate degree programs in software engineering. *Computer*, 48(11):106–109, 2015.

DANIJEL Arsenovski. Swarm: beyond pair, beyond scrum. *Experience report, Agile*, 2016.

John Baer. Is creativity domain specific? *The Cambridge handbook of creativity*, pages 321–341, 2010.

John Baer and Sharon S McKool. Assessing creativity using the consensual assessment technique. In *Handbook of research on assessment technologies, methods, and applications in higher education*, pages 65–77. IGI Global, 2009.

Janet L Bailey. Non-technical skills for success in a technical world. *International Journal of Business and Social Science*, 5(4), 2014.

Baptiste Barbot. Measuring creativity change and development. *Psychology*

of Aesthetics, Creativity, and the Arts, 13(2):203, 2019.

Baptiste Barbot, Richard W Hass, and Roni Reiter-Palmon. Creativity assessment in psychological research:(re) setting the standards. *Psychology of Aesthetics, Creativity, and the Arts*, 13(2):233, 2019.

Shaowen Bardzell, Daniela K Rosner, and Jeffrey Bardzell. Crafting quality in design: integrity, creativity, and public sensibility. In *Proceedings of the designing interactive systems conference*, pages 11–20, 2012.

William H Bares, Bill Manaris, Renée McCauley, and Christine Moore. Achieving gender balance through creative expression. In *Proceedings of the 50th SIGCSE Technical Symposium*, pages 293–299, 2019.

David John Barnes, Michael Kölling, and James Gosling. *Objects First with Java: A practical introduction using BlueJ*. Pearson Prentice Hall London, 2006.

Frank Barron and David M Harrington. Creativity, intelligence, and personality. *Annual review of psychology*, 32(1):439–476, 1981.

María Cecilia Bastarrica, Daniel Perovich, and Maíra Marques Samary. What can students get from a software engineering capstone course? In *Software Engineering: Software Engineering Education and Training Track (ICSE-SEET), 2017 IEEE/ACM 39th International Conference on*, pages 137–145. IEEE, 2017.

Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. *Agile Manifesto Online*, 2001.

Brett A Becker. What does saying that 'programming is hard' really say, and about whom? *Commun. ACM*, 64(8):27–29, jul 2021. ISSN 0001-0782. 10.1145/3469115. URL https://doi.org/10.1145/3469115.

Brett A Becker and Thomas Fitzpatrick. What do cs1 syllabi reveal about our expectations of introductory programming students? In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 1011–1017, 2019a.

Brett A Becker and Thomas Fitzpatrick. What do cs1 syllabi reveal about our expectations of introductory programming students? In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 1011–1017, New York, NY, USA, 2019b. Association for Computing Machinery. ISBN 9781450358903. 10.1145/3287324.3287485. URL https://doi.org/10.1145/3287324.3287485.

Brett A Becker and Thomas Fitzpatrick. What do cs1 syllabi reveal about our expectations of introductory programming students? In *Proceedings of the 50th SIGCSE Technical Symposium*, SIGCSE '19, page 1011–1017, NY, NY, USA, 2019c. ACM. ISBN 9781450358903.

Brett A Becker and Keith Quille. 50 years of cs1 at sigcse: A review of the evolution of introductory programming education research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science*

Education, SIGCSE '19, page 338–344, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450358903. 10.1145/3287324.3287432. URL https://doi.org/10.1145/3287324.3287432.

Ronald A Beghetto and James C Kaufman. Toward a broader conception of creativity: A case for" mini-c" creativity. *Psychology of Aesthetics, Creativity, and the Arts*, 1(2):73, 2007.

Jens Bennedssen and Michael E Caspersen. Abstraction ability as an indicator of success for learning computing science? In *Proceedings of the fourth international workshop on computing education research*, pages 15–26, 2008.

Michael Mose Biskjaer. *Self-imposed creativity constraints*. PhD thesis, Department of Aesthetics and Communication, Faculty of Arts, Aarhus University, 2013.

Michael Mose Biskjaer and Kim Halskov. Self-imposed constraints as a creative resource in art and interaction design. In *Second International Symposium on Culture, Creativity and Interaction Design, Newcastle, England. Retrieved from http://ccid2. files. wordpress. com/2011/02/biskjaer-halskov. pdf*, 2011.

Michael Mose Biskjaer, Bo T Christensen, Morten Friis-Olivarius, Sille JJ Abildgaard, Caroline Lundqvist, and Kim Halskov. How task constraints affect inspiration search strategies. *International Journal of Technology and Design Education*, 30(1):101–125, 2020.

Thomas Bjørner and Lise Busk Kofoed. How academic teachers perceive and facilitate creativity. *European Journal of Engineering Education*, 38(5):556–566, 2013.

Finn Olav Bjørnson and Torgeir Dingsøyr. Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. *Information and Software Technology*, 50(11):1055–1068, 2008.

Anna E Bobkowska. Exploration of creativity techniques in software engineering in training-application-feedback cycle. In *Workshop on Enterprise and Organizational Modeling and Simulation*, pages 99–118. Springer, 2019.

Margaret A Boden. Creativity and artificial intelligence. *Artificial intelligence*, 103(1-2):347–356, 1998.

Jürgen Börstler, Harald Störrle, Daniel Toll, Jelle van Assema, Rodrigo Duran, Sara Hooshangi, Johan Jeuring, Hieke Keuning, Carsten Kleiner, and Bonnie MacKellar. " i know it when i see it" perceptions of code quality: Iticse'17 working group report. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*, pages 70–85, 2018.

Pierre Bourque, Richard E Fairley, et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

Eric Brechner. Things they would not teach me of in college: what microsoft developers learn later. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 134–136, 2003.

Wayne Brookes. On creativity and innovation in the computing curriculum. In *Proceedings of the 20th Australasian Computing Education Conference*, pages 17–24, 2018.

Richard Buchanan. Wicked problems in design thinking. *Design issues*, 8 (2):5–21, 1992.

Luiz Fernando Capretz. Bringing the human factor to software engineering. *IEEE software*, 31(2):104–104, 2014.

Luiz Fernando Capretz and Fahem Ahmed. A call to promote soft skills in software engineering. *Psychology and Cognitive Sciences*, 4, 2018.

Valerie Carr, Morris Jones, and Belle Wei. Interdisciplinary computing: Applied computing for behavioral and social sciences. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 400–406, 2020.

Erin A Carroll, Celine Latulipe, Richard Fung, and Michael Terry. Creativity factor evaluation: towards a standardized survey metric for creativity support. In *Proceedings of the seventh ACM conference on Creativity and cognition*, pages 127–136, 2009.

Lori Carter. Interdisciplinary computing classes: Worth the effort. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, page 445–450, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450326056. 10 .1145 / 2538862 .2538882. URL https://doi-org.kuleuven.e-bronnen.be/10.1145/2538862.2538882.

Paula Catarino, Maria M Nascimento, Eva Morais, Helena Campos, and Paulo Vasco. Breaking the habit: engineering students' understanding of mathematical creativity. *European Journal of Engineering Education*, 44(4):449–460, 2019.

Katherine Cennamo, Sarah A Douglas, Mitzi Vernon, Carol Brandt, Brigitte Scott, Yolanda Reimer, and Margarita McGrath. Promoting creativity in the computer science design studio. In *Proceedings of the 42nd SIGCSE Technical Symposium*, pages 649–654, 2011.

Rosanna Yuen-Yan Chan, Cecilia Ka Yuk Chan, Carmen Lau, Jiaxin Liang, Huaiyi Huang, Qi Cao, and Mehrdad Tahernia. Direct evidence of engineering students. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–5. IEEE, 2017.

Hadas Chassidim, Dani Almog, and Shlomo Mark. Fostering soft skills in project-oriented learning within an agile atmosphere. *European Journal of Engineering Education*, 43(4):638–650, 2018.

Robert Chatley and Tony Field. Lean learning-applying lean techniques to improve software engineering education. In *2017 IEEE/ACM 39th*

International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET), pages 117–126. IEEE, 2017.

Kuanchin Chen and Alan Rea. Do pair programming approaches transcend coding? measuring agile attitudes in diverse information systems courses. *Journal of Information Systems Education*, 29(2):53–64, 2018.

Lorenza S Colzato, Ayca Szapora Ozturk, Justine Nienke Pannekoek, and Bernhard Hommel. The impact of physical exercise on convergent and divergent thinking. *Frontiers in human neuroscience*, page 824, 2013.

Paul T Costa Jr and Robert R McCrae. *The Revised Neo Personality Inventory (neo-pi-r)*. Sage Publications, Inc, 2008.

Sofie Craps, Maarten Pinxten, Heidi Knipprath, and Greet Langie. Different roles, different demands. a competency-based professional roles model for early career engineers, validated in industry and higher education. *European Journal of Engineering Education*, 47(1):144–163, 2022.

Genevieve M Cseh and Karl K Jeffries. A scattered cat: A critical evaluation of the consensual assessment technique for creativity research. *Psychology of Aesthetics, Creativity, and the Arts*, 13(2):159, 2019.

Mihaly Csikszentmihalyi. Flow and the psychology of discovery and invention. *HarperPerennial, New York*, 39, 1997.

Mihaly Csikszentmihalyi. Society, culture, and person: A systems view of creativity. In *The systems model of creativity*, pages 47–61. Springer, 2014.

Uriel Rubén Cukierman and Juan María Palmieri. Soft skills in engineering education: A practical experience in an undergraduate course. In *Interactive Collaborative Learning (ICL), 2014 International Conference on*, pages 237–242. IEEE, 2014.

Jacob Cybulski, Lemai Nguyen, Theerasak Thanasankit, and Sharman Lichtenstein. Understanding problem solving in requirements engineering: Debating creativity with is practitioners. In *PACIS 2003: Proceedings of the Seventh Pacific Asia Conference on Information Systems*, pages 465–482. University of South Australia, 2003.

Claudio León de la Barra, Broderick Crawford, Ricardo Soto, Sanjay Misra, and Eric Monfroy. Agile software development: It is about knowledge management and creativity. In *International Conference on Computational Science and Its Applications*, pages 98–113. Springer, 2013.

Cameron D Denson, Jennifer K Buelin, Matthew D Lammi, and Susan D'Amico. Developing instrumentation for assessing creativity in engineering design. *Journal of Technology Education*, 27(1):23–40, 2015.

Stuart E Dreyfus and Hubert L Dreyfus. A five-stage model of the mental activities involved in directed skill acquisition. Technical report,

California Univ Berkeley Operations Research Center, 1980.

Angela L Duckworth, Christopher Peterson, Michael D Matthews, and Dennis R Kelly. Grit: perseverance and passion for long-term goals. *Journal of personality and social psychology*, 92(6):1087, 2007.

Joanna C Dunlap and Scott Grabinger. Preparing students for lifelong learning: A review of instructional features and teaching methodologies. *Performance Improvement Quarterly*, 16(2):6–25, 2003.

Alessandra CS Dutra, Rafael Prikladnicki, and César França. What do we know about high performance teams in software engineering? results from a systematic literature review. In *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on*, pages 183–190. IEEE, 2015.

Arne Duyver. De invloed van interdisciplinaire samenwerking tussen software-ingenieur- en designstudenten op de creativiteit. Master's thesis, Faculty of Engineering Technology, KU Leuven and UHasselt, 2022.

Arne Duyver, Wouter Groeneveld, and Kris Aerts. Does interdisciplinary creative coding boost creativity? a mixed methods approach. In *Proceedings of the 51th Annual SEFI Conference 2023*, in press.

Carol S Dweck. *Mindset: The new psychology of success*. Random House Digital, Inc., 2008.

Robert L Ebel and David A Frisbie. *Essentials of educational measurement*. Prentice-Hall Englewood Cliffs, NJ, 1972.

Andreas W Ebentheuer, Jörg Kammermann, and Hans-Georg Herzog. Adveisor—analysis of an established soft skill program for students in the field of electrical engineering. In *Global Engineering Education Conference (EDUCON), 2017 IEEE*, pages 468–475. IEEE, 2017.

Umberto Eco et al. *The open work*. Harvard University Press, 1989.

Robert Epstein, Steven M Schmidt, and Regina Warfel. Measuring and training creativity competencies: Validation of a new test. *Creativity Research Journal*, 20(1):7–12, 2008.

Hamidreza Eskandari, Serge Sala-Diakanda, Sandra Furterer, Luis Rabelo, Lesia Crumpton-Young, and Kent Williams. Enhancing the undergraduate industrial engineering curriculum: Defining desired characteristics and emerging topics. *Education+ Training*, 49(1):45–55, 2007.

Marisa Exter, Secil Caskurlu, and Todd Fernandez. Comparing computing professionals' perceptions of importance of skills and knowledge on the job and coverage in undergraduate experiences. *ACM Transactions on Computing Education (TOCE)*, 18(4):1–29, 2018.

Peter A Facione. The disposition toward critical thinking: Its character, measurement, and relationship to critical thinking skill. *Informal logic*, 20(1), 2000.

Nancy Falchikov and Judy Goldfinch. Student peer assessment in higher

education: A meta-analysis comparing peer and teacher marks. *Review of educational research*, 70(3):287–322, 2000.

Stephen Fallows and Balasubramanyan Chandramohan. Multiple approaches to assessment: Reflections on use of tutor, peer and self-assessment. *Teaching in Higher Education*, 6(2):229–246, 2001.

Edward F Fern and Edward E Fern. *Advanced focus group research*. Sage, 2001.

José Figueiredo, Noel Lopes, and Francisco José García-Peñalvo. Predicting student failure in an introductory programming course with multiple back-propagation. In *Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality*, pages 44–49, 2019.

Ian Finlayson. Using the game boy advance to teach computer systems and architecture. *Journal of Computing Sciences in Colleges*, 32(3):78–84, 2017.

Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

Batya Friedman and Peter H Kahn. Educating computer scientists: linking the social and the technical. *Commun. ACM*, 37(1):64–70, 1994.

Keston H Fulcher. *Towards measuring lifelong learning: The curiosity index*. PhD thesis, ProQuest Information & Learning, 2004.

Ursula Fuller, Arnold Pears, June Amillo, Chris Avram, and Linda Mannila. A computing perspective on the bologna process. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '06, pages 115–131, New York, NY, USA, 2006. ACM. ISBN 1-59593-603-3. 10.1145/1189215.1189181. URL http://doi.acm.org/10.1145/1189215.1189181.

Matthias Galster, Antonija Mitrovic, and Matthew Gordon. Toward enhancing the training of software engineering students and professionals using active video watching. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*, pages 5–8. ACM, 2018.

Vahid Garousi, Görkem Giray, Eray Tüzün, Cagatay Catal, and Michael Felderer. Closing the gap between software engineering education and industrial needs. *arXiv preprint arXiv:1812.01954*, 2018.

Vahid Garousi, Gorkem Giray, and Eray Tuzun. Understanding the knowledge gaps of software engineers: an empirical analysis based on swebok. *ACM Transactions on Computing Education (TOCE)*, 20(1): 1–33, 2019a.

Vahid Garousi, Gorkem Giray, Eray Tuzun, Cagatay Catal, and Michael Felderer. Closing the gap between software engineering education and industrial needs. *IEEE Software*, 37(2):68–77, 2019b.

John S Gero. Computational models of innovative and creative design processes. *Technological forecasting and social change*, 64(2-3):183–196, 2000.

Vlad Petre Glaveanu, Michael Hanchett Hanson, John Baer, Baptiste Barbot, Edward P Clapp, Giovanni Emanuele Corazza, Beth Hennessey, James C Kaufman, Izabela Lebuda, Todd Lubart, et al. Advancing creativity theory and research: A socio-cultural manifesto. *The Journal of Creative Behavior*, 54(3):741–745, 2020.

Maarten Goossens and Sven De Maeyer. How to obtain efficient high reliabilities in assessing texts: Rubrics vs comparative judgement. In *International Conference on Technology Enhanced Assessment*, pages 13–25. Springer, 2017.

Timo Göttel and Jonas Schild. Creativity room 5555: Evoking creativity in game design amongst cs students. In *ITiCSE 2011*, pages 98–102, 2011.

Nuno Gracias and Rafael Garcia. Can we trust peer grading in oral presentations? towards optimizing a critical resource nowadays: Teachers' time. In *5th International Conference on Education and New Learning Technologies (EDULEARN), Barcelona, Spain. Retrieved from http://users. isr. ist. utl. pt/~ ngracias/publications/Gracias13_Edulearn_1329. pdf*, 2013.

Daniel Graziotin, Xiaofeng Wang, and Pekka Abrahamsson. Happy software developers solve problems better: psychological measurements in empirical software engineering. *PeerJ*, 2:e289, 2014.

Daniel Graziotin, Xiaofeng Wang, and Pekka Abrahamsson. Understanding the affect of developers: theoretical background and guidelines for psychoempirical software engineering. In *Proceedings of the 7th International Workshop on Social Software Engineering*, pages 25–32. ACM, 2015.

Ira Greenberg, Deepak Kumar, and Dianna Xu. Creative coding and visual portfolios for cs1. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 247–252, 2012.

Wouter Groeneveld. *The Creative Programmer*. Manning, 2023. ISBN 9781633439054.

Wouter Groeneveld and Kris Aerts. Sparking creativity with the game boy advance. In *Proceeding of the 51th ACM technical symposium on computer science education*. Association for Computing Machinery (ACM), 2020.

Wouter Groeneveld, Joost Vennekens, and Kris Aerts. Software engineering education beyond the technical: A systematic literature review. In *Proceedings of the 47th Annual SEFI Conference 2019*, pages 1607–1622. European Society for Engineering Education SEFI, 2019.

Wouter Groeneveld, Brett A Becker, and Joost Vennekens. 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*, pages 287–293, 2020a.

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

Wouter Groeneveld, Joost Vennekens, and Kris Aerts. Engaging software engineering students in grading: The effects of peer assessment on self-evaluation, motivation, and study time. In *Proceedings of the 48th SEFI Conference 2020*, volume 48, pages 799–809. European Society for Engineering Education SEFI, 2020c.

Wouter Groeneveld, Laurens Luyten, Joost Vennekens, and Kris Aerts. 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, 2021a.

Wouter Groeneveld, Joost Vennekens, and Kris Aerts. Identifying non-technical skill gaps in software engineering education: What experts expect but students don't learn. *ACM Transactions on Computing Education (TOCE)*, 22(1):1–21, 2021b.

Wouter Groeneveld, Brett A Becker, and Joost Vennekens. How creatively are we teaching and assessing creativity in computing education: A systematic literature review. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, pages 934–940, 2022a.

Wouter Groeneveld, Dries Martin, Tibo Poncelet, and Kris Aerts. Are undergraduate creative coders clean coders? a correlation study. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, pages 314–320, 2022b.

Wouter Groeneveld, Lynn Van den Broeck, Joost Vennekens, and Kris Aerts. Self-assessing creative problem solving for aspiring software developers: A pilot study. In *Proceedings of the 2022 ACM Conference on Innovation and Technology in Computer Science Education*, pages 5–11, 2022c.

Wouter Groeneveld, Laurens Luyten, Joost Vennekens, and Kris Aerts. Students' and professionals' perceived creativity in software engineering: A comparative study. *European Journal of Engineering Education*, in press.

Aleksandra Gruszka and Min Tang. The 4p's creativity model and its application in different fields. In *Handbook of the management of creativity and innovation: Theory and practice*, pages 51–71. WSPC, 2017.

Greg Guest, Emily Namey, and Mario Chen. A simple method to assess and report thematic saturation in qualitative research. *PLoS One*, 15 (5):e0232076, 2020.

Philip Guo. Python is now the most popular introductory teaching language at top US universities. *Communications of the ACM Blog (BLOG@CACM), July*, 2014.

Orit Hazzan and Gadi Har-Shai. Teaching computer science soft skills as

soft concepts. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 59–64. ACM, 2013.

Peter Johan Heiberg Engel. Tacit knowledge and visual expertise in medical diagnostic reasoning: implications for medical education. *Medical teacher*, 30(7):e184–e188, 2008.

Philipp Holtkamp, Jussi PP Jokinen, and Jan M Pawlowski. Soft competency requirements in requirements engineering, software design, implementation, and testing. *Journal of Systems and Software*, 101:136–146, 2015.

Adrian Holzer, Samuel Bendahan, Isabelle Vorneche Cardia, and Denis Gillet. Early awareness of global issues and development of soft skills in engineering education: An interdisciplinary approach to communication. In *Information Technology Based Higher Education and Training (ITHET), 2014*, pages 1–6. IEEE, 2014.

Siw Elisabeth Hove and Bente Anda. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05)*, pages 10–pp. IEEE, 2005.

Michael JA Howe, Jane W Davidson, and John A Sloboda. Innate talents: Reality or myth? *Behavioral and brain sciences*, 21(3):399–407, 1998.

Andrew Hunt. *Pragmatic thinking and learning: Refactor your Wetware*. Pragmatic bookshelf, 2008.

IEEE Computer Society. *IEEE Standard Glossary of Software Engineering Terminology*. IEEE, 1990.

Ville Isomöttönen, Mats Daniels, Åsa Cajander, Arnold Pears, and Roger Mcdermott. Searching for global employability: Can students capitalize on enabling learning environments? *ACM Transactions on Computing Education (TOCE)*, 19(2):1–29, 2019.

Susan A Jackson and Herbert W Marsh. Development and validation of a scale to measure optimal experience: The flow state scale. *Journal of sport and exercise psychology*, 18(1):17–35, 1996.

Ivar Jacobson and Ed Seidewitz. A new software engineering. *Communications of the ACM*, 57(12):49–54, 2014.

Mary Elizabeth ME Jones, Melanie Kisthardt, and Marie A Cooper. Interdisciplinary teaching: Introductory programming via creative writing. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 523–528, 2011.

Anna Jordanous. Creativity vs quality: why the distinction matters when evaluating computational creativity systems. In *Proceedings of The 5th Computational Creativity Symposium at the AISB Convention*. AISB, 2018.

Henry F Kaiser. An index of factorial simplicity. *Psychometrika*, 39(1):31–36, 1974.

Ruth Katz. Collective" problem-solving" in the history of music: The case of the camerata. *Journal of the History of Ideas*, pages 361–377, 1984.

Jen Katz-Buonincontro, Richard W Hass, and Gary Friedman. "engineering" student creativity in a probability and statistics course: Investigating perceived versus actual creativity. *Psychology of Aesthetics, Creativity, and the Arts*, 11(3):295, 2017.

James C Kaufman. Counting the muses: development of the kaufman domains of creativity scale (k-docs). *Psychology of Aesthetics, Creativity, and the Arts*, 6(4):298, 2012.

James C Kaufman and John Baer. Beyond new and appropriate: Who decides what is creative? *Creativity Research Journal*, 24(1):83–91, 2012.

James C Kaufman and Ronald A Beghetto. Beyond big and little: The four c model of creativity. *Review of general psychology*, 13(1):1–12, 2009.

James C Kaufman and Robert J Sternberg. Creativity. *Change: The Magazine of Higher Learning*, 39(4):55–60, 2007.

James C Kaufman, Claudia A Gentile, and John Baer. Do gifted student writers and creative writing experts rate creativity the same way? *Gifted Child Quarterly*, 49(3):260–265, 2005.

Mark Keil, Hyung Koo Lee, and Tianjie Deng. Understanding the most critical skills for managing it projects: A delphi study of it project managers. *Information & management*, 50(7):398–414, 2013.

Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. Code quality issues in student programs. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, pages 110–115, 2017.

Jin-Young Kim. A longitudinal study of the relation between creative potential and academic achievement at an engineering university in korea. *Journal of Engineering Education*, 109(4):704–722, 2020.

John R Kirby, Christopher Knapper, Patrick Lamon, and William J Egnatoff. Development of a scale to measure lifelong learning. *International Journal of Lifelong Education*, 29(3):291–302, 2010.

Barbara Ann Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 07 2007. URL https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf.

Rex B Kline. *Principles and practice of structural equation modeling*. Guilford publications, 2015.

Maria Knobelsdorf and Ralf Romeike. Creativity as a pathway to computer science. In *ITiCSE 2008*, pages 286–290, 2008.

Jyrki Kontio, Laura Lehtola, and Johanna Bragge. Using the focus group method in software engineering: obtaining practitioner and user experiences. In *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE'04.*, pages 271–280. IEEE, 2004.

Özgen Korkmaz, Recep Çakir, and M Yaşar Özden. A validity and reliability

study of the computational thinking scales (cts). *Computers in human behavior*, 72:558–569, 2017.

Andreas Krapp. Interest, motivation and learning: An educational-psychological perspective. *European journal of psychology of education*, 14(1):23–40, 1999.

Stephan Krusche, Dora Dzvonyar, Han Xu, and Bernd Bruegge. Software theater—teaching demo-oriented prototyping. *ACM Transactions on Computing Education (TOCE)*, 18(2):1–30, 2018.

Hsu-Chan Kuo. Toward a synthesis framework for the study of creativity in education: an initial attempt. *Educate~*, 11(1):65–75, 2011.

Per Lenberg, Robert Feldt, and Lars Göran Wallgren. Behavioral software engineering: A definition and systematic literature review. *Journal of Systems and software*, 107:15–37, 2015.

Timothy C Lethbridge, Richard J LeBlanc Jr, Ann E Kelley Sobel, Thomas B Hilburn, and Jorge L Diaz-Herrera. Se2004: Recommendations for undergraduate software engineering curricula. *IEEE software*, 23(6): 19–25, 2006.

Meira Levy. Educating for empathy in software engineering course. In *Joint Proceedings of REFSQ-2018 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track*. CEUR, 2018.

Kin Fun Li, John Fagan, and Imen Bourguiba. Teaching professional practice and career development to graduate students. In *Teaching, Assessment, and Learning for Engineering (TALE), 2016 IEEE International Conference on*, pages 398–402. IEEE, 2016.

Paul Luo Li, Amy J Ko, and Jiamin Zhu. What makes a great software engineer? In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 700–710. IEEE, 2015.

Janet Liebenberg, Magda Huisman, and Elsa Mentz. Industry's perception of the relevance of software development education. *TD: The Journal for Transdisciplinary Research in Southern Africa*, 11(3):260–284, 2015.

James J Lu and George HL Fletcher. Thinking about computational thinking. In *Proceedings of the 40th SIGCSE Technical Symposium*, pages 260–264, 2009.

Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. Introductory programming: A systematic literature review. In *ITiCSE 2018*, ITiCSE 2018 Companion, page 55–106, NY, NY, USA, 2018. ACM. ISBN 9781450362238.

Maira Marques, Sergio F Ochoa, Maria Cecilia Bastarrica, and Francisco J Gutierrez. Enhancing the student learning experience in software engineering project courses. *IEEE Transactions on Education*, 61(1): 63–73, 2018.

Bryan Marshall, Peter Cardon, Amit Poddar, and Renee Fontenot. Does

sample size matter in qualitative research?: A review of qualitative interviews in is research. *Journal of computer information systems*, 54 (1):11–22, 2013.

Robert C Martin. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

Catalina Martinez-Mediano and Susan M Lord. Lifelong learning competencies program for engineers. *International Journal of Engineering Education*, 28(1):130, 2012.

Miriam L Matteson, Lorien Anderson, and Cynthia Boyden. "soft skills": A phrase in search of meaning. *portal: Libraries and the Academy*, 16(1): 71–88, 2016.

Gerardo Matturro, Florencia Raschetti, and Carina Fontán. Soft skills in software development teams: A survey of the points of view of team leaders and team members. In *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 101–104. IEEE Press, 2015.

Bruce R Maxim, Stein Brunvand, and Adrienne Decker. Use of role-play and gamification in a software project course. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–5. IEEE, 2017.

A McAlister, D Lee, KM Ehlert, RL Kajfez, CJ Faber, and MS Kennedy. Qualitative coding: An approach to assess inter-rater reliability. In *Columbus, Ohio https://peer asee org/28777: ASEE Annual Conference & Exposition*, 2017.

Philippe Méhaut and Christopher Winch. The european qualification framework: skills, competences or knowledge? *European educational research journal*, 11(3):369–381, 2012.

Martin Meinel, Lukas Maier, Timm Wagner, and Kai-Ingo Voigt. Designing creativity-enhancing workspaces: A critical look at empirical evidence. *Journal of technology and innovation management*, 1(1), 2017.

Angie L Miller. A self-report measure of cognitive processes associated with creativity. *Creativity Research Journal*, 26(2):203–218, 2014a.

Lee Dee Miller, Leen-Kiat Soh, Vlad Chiriacescu, Elizabeth Ingraham, Duane F Shell, and Melissa Patterson Hazley. Integrating computational and creative thinking to improve learning and performance in cs1. In *Proceedings of the 45th SIGCSE Technical Symposium*, pages 475–480, 2014.

Steven Miller. Collaborative approaches needed to close the big data skills gap. *Journal of Organization design*, 3(1):26–30, 2014b.

Jan L Mize. Making an academic curriculum relevant to business requirements. *ACM SIGCSE Bulletin*, 8(2):24–27, 1976.

Rahul Mohanani, Prabhat Ram, Ahmed Lasisi, Paul Ralph, and Burak Turhan. Perceptions of creativity in software engineering research and practice. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 210–217. IEEE, 2017.

Catherine Mooney and Brett A Becker. Investigating the impact of the covid-19 pandemic on computing students' sense of belonging. In *Proceedings of the 52nd SIGCSE Technical Symposium*, SIGCSE '21, page 612–618, NY, NY, USA, 2021. ACM. ISBN 9781450380621. 10 .1145/3408877.3432407. URL https://doi.org/10.1145/3408877.3432407.

Ross L Mooney. A conceptual model for integrating four approaches to the identification of creative talent. *Scientific creativity: Its recognition and development*, pages 331–340, 1963.

Ana M Moreno, Maria-Isabel Sanchez-Segura, Fuensanta Medina-Dominguez, and Laura Carvajal. Balancing software engineering education and industrial needs. *Journal of systems and software*, 85(7): 1607–1620, 2012.

NJ Mouros. Defining, teaching and assessing lifelong learning skills. In *33rd Annual Frontiers in Education, 2003. FIE 2003.*, volume 1, pages T3B–14. IEEE, 2003.

Amir Mujkanovic and Andreas Bollin. Improving learning outcomes through systematic group reformation: The role of skills and personality in software engineering education. In *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 97–103. ACM, 2016.

Alla Nerubasska and Borys Maksymchuk. The demarkation of creativity, talent and genius in humans: a systemic aspect. *Postmodern Openings/Deschideri Postmoderne*, 11(2), 2020.

Cal Newport. *Deep work: Rules for focused success in a distracted world*. Hachette UK, 2016.

Lemai Nguyen and Graeme Shanks. A framework for understanding creativity in requirements engineering. *Information and software technology*, 51(3):655–662, 2009.

Moti Nissani. Ten cheers for interdisciplinarity: The case for interdisciplinary knowledge and research. *The social science journal*, 34(2):201–216, 1997.

Ed OECD. The future of education and skills: Education 2030. *OECD Education Working Papers*, pages 1–23, 2018.

Kathleen Ofstedal and Kathryn Dahlberg. Collaboration in student teaching: Introducing the collaboration self-assessment tool. *Journal of Early Childhood Teacher Education*, 30(1):37–48, 2009.

Preston Oihus, Andrea Surovek, and Dean Jensen. Design wars: Developing student creativity through competition. In *2013 IEEE Frontiers in Education Conference (FIE)*, pages 137–139. IEEE, 2013.

Chitu Okoli and Suzanne D Pawlowski. The delphi method as a research tool: an example, design considerations and applications. *Information & management*, 42(1):15–29, 2004.

Anthony J Onwuegbuzie, Wendy B Dickinson, Nancy L Leech, and

Annmarie G Zoran. A qualitative framework for collecting and analyzing data in focus group research. *International journal of qualitative methods*, 8(3):1–21, 2009.

Eleonora Papaleontiou-Louca, Despina Varnava-Marouchou, Simona Mihai, and Elmos Konis. Teaching for creativity in universities. *Journal of Education and Human Development*, 3(4):131–154, 2014.

Marina Papastergiou. Digital Game-Based Learning in high school Computer Science education: Impact on educational effectiveness and student motivation. *Computers and Education*, 52(1):1–12, 2009. ISSN 03601315. 10.1016/j.compedu.2008.06.004. URL http://dx.doi.org/10.1016/j.compedu.2008.06.004.

Frederick Patacsil and Christine Lourrine S Tablatin. Exploring the importance of soft and hard skills as perceived by it internship students and industry: A gap analysis. *JOTSE*, 7(3):347–368, 2017.

K Peppler and Y Kafai. Creative coding: Programming for personal expression. *Retrieved August*, 30(2008):314, 2005.

Markeya S Peteranetz, Shiyuan Wang, Duane F Shell, Abraham E Flanigan, and Leen-Kiat Soh. Examining the impact of computational creativity exercises on college computer science students' learning, achievement, self-efficacy, and creativity. In *Proceedings of the 49th SIGCSE Technical Symposium*, pages 155–160, 2018.

Markeya S Peteranetz, Leen-Kiat Soh, and Elizabeth Ingraham. Building computational creativity in an online course for non-majors. In *Proceedings of the 50th SIGCSE Technical Symposium*, pages 442–448, 2019.

ED Petkus Jr. The creative identity: Creative behavior from the symbolic interactionist perspective. *The Journal of Creative Behavior*, 30(3):188–196, 1996.

James Prather, Brett A Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. What do we think we think we are doing? metacognition and self-regulation in programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, pages 2–13, 2020.

Alex Radermacher and Gursimran Walia. Gaps between industry expectations and the abilities of graduates. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 525–530. ACM, 2013.

Alex Radermacher, Gursimran Walia, and Dean Knudson. Investigating the skill gap between graduating students and industry expectations. In *Companion Proceedings of the 36th international conference on software engineering*, pages 291–300, 2014.

Martin Reuter, J Panksepp, N Schnabel, N Kellerhoff, P Kempel, and J Hennig. Personality and biological markers of creativity. *European Journal of Personality: Published for the European Association of*

Personality Psychology, 19(2):83–95, 2005.

Mel Rhodes. An analysis of creativity. *The Phi Delta Kappan*, 42(7):305–310, 1961.

Lauren Rich, Heather Perry, and Mark Guzdial. A cs1 course designed to address interests of women. *ACM SIGCSE Bulletin*, 36(1):190–194, 2004.

K Andrew R Richards and Michael A Hemphill. A practical guide to collaborative qualitative data analysis. *Journal of Teaching in Physical Education*, 37(2):225–231, 2018.

Simone M Ritter, Madelijn Strick, Maarten W Bos, Rick B Van Baaren, and AP Dijksterhuis. Good morning creativity: task reactivation during sleep enhances beneficial effect of sleep on creative performance. *Journal of sleep research*, 21(6):643–647, 2012.

Letícia Rodrigues da Fonseca, Marcelo Ribeiro Silva, Sheldon William Silva, and Guilherme Marques Pereira. Continuous-learning work environment: A study with developers in software development organizations. *Knowledge Management & E-Learning*, 11(3):281–303, 2019.

Pilar Rodríguez, Jari Partanen, Pasi Kuvaja, and Markku Oivo. Combining lean thinking and agile methods for software development: A case study of a finnish provider of wireless embedded systems detailed. In *2014 47th Hawaii International Conference on System Sciences*, pages 4770–4779. IEEE, 2014.

Ralf Romeike. Three drivers for creativity in computer science education. *Proc of Informatics, Mathematics and ICT: A 'Golden Triangle'. Boston, USA*, 2007.

Mark A Runco. "big c, little c" creativity as a false dichotomy: Reality is not categorical. *Creativity Research Journal*, 26(1):131–132, 2014a.

Mark A Runco. *Creativity: Theories and Themes: Research, Development, and Practice*. Elsevier, 2014b.

Kevin Ryan. We should teach our students what industry doesn't want. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 103–106. IEEE, 2020.

Philip M Sadler and Eddie Good. The impact of self-and peer-grading on student learning. *Educational assessment*, 11(1):1–31, 2006.

Mehdi SM Sajjadi, Morteza Alamgir, and Ulrike von Luxburg. Peer grading in a course on algorithms and data structures: Machine learning algorithms do not improve over simple baselines. In *Proceedings of the third (2016) ACM conference on Learning@ Scale*, pages 369–378, 2016.

Andrea Salgian, Teresa M Nakra, Christopher Ault, and Yunfeng Wang. Teaching creativity in computer science. In *Proceeding of the 44th SIGCSE Technical Symposium*, pages 123–128, 2013.

Kate Sanders, Judy Sheard, Brett A Becker, Anna Eckerdal, and Sally

Hamouda. Inferential statistics in computing education research: A methodological review. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 177–185, 2019.

Michael Schumm, Saskia Joseph, Irmgard Schroll-Decker, Michael Niemetz, and Jürgen Mottok. Required competences in software engineering: Pair programming as an instrument for facilitating life-long learning. In *2012 15th International Conference on Interactive Collaborative Learning (ICL)*, pages 1–5. IEEE, 2012.

Yvonne Sedelmaier and Dieter Landes. Practicing soft skills in software engineering: A project-based didactical approach. In *Overcoming Challenges in Software Engineering Education: Delivering Non-Technical Knowledge and Skills*, pages 161–179. IGI Global, 2014a.

Yvonne Sedelmaier and Dieter Landes. Software engineering body of skills (swebos). In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 395–401. IEEE, 2014b.

Jacqueline Senker. Tacit knowledge and models of innovation. *Industrial and corporate change*, 4(2):425–447, 1995.

Duane F Shell, Leen-Kiat Soh, Abraham E Flanigan, Markeya S Peteranetz, and Elizabeth Ingraham. Improving students' learning and achievement in cs classrooms through computational creativity exercises that integrate computational and creative thinking. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 543–548, 2017.

William R Sherrard, Feraidoon Raafat, and Richard R Weaver. An empirical study of peer bias in evaluations: Students rating students. *Journal of Education for Business*, 70(1):43–47, 1994.

Ben Shneiderman. Creating creativity: User interfaces for supporting innovation. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):114–138, 2000.

Larry J Shuman, Mary Besterfield-Sacre, and Jack McGourty. The abet "professional skills"—can they be taught? can they be assessed? *Journal of engineering education*, 94(1):41–55, 2005.

Paul J Silvia, James C Kaufman, and Jean E Pretz. Is creativity domain-specific? latent class models of creative accomplishments and creative self-descriptions. *Psychology of Aesthetics, Creativity, and the Arts*, 3(3):139, 2009.

Dean Keith Simonton. Creativity: Cognitive, personal, developmental, and social aspects. *American psychologist*, 55(1):151, 2000.

Dean Keith Simonton. Does talent exist?: Yes! In *Routledge handbook of talent identification and development in sport*, pages 10–18. Routledge, 2017.

Jirarat Sitthiworachart and Mike Joy. Effective peer assessment for learning computer programming. *ACM SIGCSE Bulletin*, 36(3):122–126, 2004.

Dominique Sluijsmans, Filip Dochy, and George Moerkerke. Creating a learning environment by using self-, peer-and co-assessment.

Learning environments research, 1(3):293–319, 1998.

John Z Sonmez. *Soft skills: The Software Developer's Life Manual*. Manning Publications,, 2016.

Alexander Soska, Jürgen Mottok, and Christian Wolff. Playful learning in academic software engineering education. In *Global Engineering Education Conference (EDUCON), 2015 IEEE*, pages 324–332. IEEE, 2015.

Edward M Sosu. The development and psychometric validation of a critical thinking disposition scale. *Thinking skills and creativity*, 9:107–119, 2013.

Zlatko Stapić, Eva García López, Antonio García Cabot, Luis de Marcos Ortega, and Vjeran Strahonja. Performing systematic literature review in software engineering. In *CECIIS 2012-23rd International Conference*, 2012.

Lorraine AJ Stefani. Assessment in partnership with learners. *Assessment & Evaluation in Higher Education*, 23(4):339–350, 1998.

Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. Designing a rubric for feedback on code quality in programming courses. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, pages 160–164, 2016.

Robert J Sternberg and Todd Lubart. An investment theory of creativity and its development. *Human development*, 34(1):1–31, 1991.

Matt Stevens and Richard Norman. Industry expectations of soft skills in it graduates: A regional survey. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '16, pages 13:1–13:9, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4042-7. 10.1145/2843043 .2843068. URL http://doi.acm.org/10.1145/2843043.2843068.

Angus Stevenson. *Oxford dictionary of English*. Oxford University Press, USA, 2010.

Jeff Sutherland and Hugo Heitz. Scrum and lean: How a lean scrum can improve your performance. Technical report, Scrum Inc, 2011.

Daymy Tamayo Avila, Wim Van Petegem, Y Cruz Ochoa, and M Noda Hernández. A correlational study on factors that influence the cohesion of software engineering students teams. In *INTED2018 Proceedings*, pages 5697–5702. IATED, 2018.

Vicki N Tariq, Eileen M Scott, A Clive Cochrane, Maria Lee, and Linda Ryles. Auditing and mapping key skills within university curricula. *Quality Assurance in Education*, 2004.

Bruce Tate. *Seven Languages in Seven Weeks: a pragmatic guide to learning programming languages*. Pragmatic Bookshelf, 2010.

Roselyne Tchoua, Jong Choi, Scott Klasky, Qing Liu, Jeremy Logan, Kenneth Moreland, Jingqing Mu, Manish Parashar, Norbert Podhorszki, David Pugmire, et al. Adios visualization schema: A first

step towards improving interdisciplinary collaboration in high performance computing. In *2013 IEEE 9th International Conference on e-Science*, pages 27–34. IEEE, 2013.

The IEEE CS/ACM Joint Task Force on Computing Curricula. Computing curricula 2001, computer science. Technical report, IEEE/ACM, 2001. URL https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2001.pdf.

The IEEE CS/ACM Joint Task Force on Computing Curricula. Computer science curriculum 2008: An interim revision of cs 2001. Technical report, IEEE/ACM, 2008. URL https://www.acm.org/binaries/content/assets/education/curricula-recommendations/computerscience2008.pdf.

The IEEE CS/ACM Joint Task Force on Computing Curricula. Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science. Technical report, IEEE/ACM, NY, NY, USA, 2013. 999133.

The IEEE CS/ACM Joint Task Force on Computing Curricula. Computing curricula 2020: Paradigms for global computing education. Technical report, IEEE/ACM, NY, NY, USA, 2020. URL https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf.

Veronika Thurner, Axel Bottcher, and Andreas Kamper. Identifying base competencies as prerequisites for software engineering education. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 1069–1076. IEEE, 2014.

Veronika Thurner, Kathrin Schlierkamp, Axel Böttcher, and Daniela Zehetmeier. Integrated development of technical and base competencies: Fostering reflection skills in software engineers to be. In *Global Engineering Education Conference (EDUCON), 2016 IEEE*, pages 340–348. IEEE, 2016.

E Paul Torrance. Predictive validity of the torrance tests of creative thinking. *The Journal of creative behavior*, 1972.

Kerrie Unsworth. Unpacking creativity. *Academy of Management Review*, 26 (2):289–297, 2001.

Klaus K Urban. Toward a componential model of creativity. *Creative intelligence: Toward theoretic integration*, pages 81—112, 2003.

Tammy VanDeGrift. Supporting creativity and user interaction in cs 1 homework assignments. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 54–59, 2015.

Tony Veale, Pablo Gervás, and Alison Pease. Understanding creativity: A computational perspective. *New Generation Computing*, 24(3):203–207, 2006.

Joost Vennekens. Service-learning for web technology: Observations from a small case study. In *ITiCSE 2020*, pages 328–334, 2020.

Aileen Vicente, Tiffany Tan, and Alvin Yu. Collaborative approach in software engineering education: An interdisciplinary case. *Journal of Information Technology Education: Innovations in Practice*, 17(1):127–152, 2018.

Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Jaakko Kurhila. Massive increase in eager tas: Experiences from extreme apprenticeship-based cs1. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 123–128, 2013.

Sophie Von Stumm, Benedikt Hell, and Tomas Chamorro-Premuzic. The hungry mind: Intellectual curiosity is the third pillar of academic performance. *Perspectives on Psychological Science*, 6(6):574–588, 2011.

Graham Wilson. Building a new mythology: The coding boot-camp phenomenon. *ACM Inroads*, 8(4):66–71, 2017.

Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, page 38. ACM, 2014.

Judy L Wynekoop and Diane B Walz. Investigating traits of top performing software developers. *Information Technology & People*, 13(3):186–195, 2000.

Dianna Xu, Ursula Wolz, Deepak Kumar, and Ira Greenberg. Updating introductory computer science with creative computation. In *Proceedings of the 49th SIGCSE Technical Symposium*, SIGCSE '18, page 167–172, NY, NY, USA, 2018. ACM. ISBN 9781450351034.

Heng-Li Yang and Hsiu-Hua Cheng. Creativity of student information system projects: From the perspective of network embeddedness. *Computers & Education*, 54(1):209–221, 2010.

Michal Young and Stuart Faulk. Sharing what we know about software engineering. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 439–442. ACM, 2010.

Timothy T Yuen. Novices' knowledge construction of difficult concepts in cs1. *ACM SIGCSE Bulletin*, 39(4):49–53, 2007.

Guangzhi Zheng, Chi Zhang, and Lei Li. Practicing and evaluating soft skills in it capstone projects. In *Proceedings of the 16th Annual Conference on Information Technology Education*, SIGITE '15, pages 109–113, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3835-6. 10.1145/2808006.2808041. URL http://doi.acm.org/10.1145/2808006.2808041.



Faculty of Engineering Technology
Engineering Technology Education Research
W. de Croylaan 56 – E box 2203
BE-3001 Heverlee, Belgium
tel. + 32 16 32 00800
wouter.groeneveld@kuleuven.be
https://iiw.kuleuven.be/onderzoek/ether