Identifying Non-Technical Skill Gaps in Software Engineering Education: What Experts Expect But Teachers Don't Teach

WOUTER GROENEVELD, KU Leuven, Belgium
JOOST VENNEKENS, KU Leuven, Belgium
KRIS AERTS, KU Leuven, Belgium

As the importance of non-technical skills in the software engineering 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 three previous works, we identify the three biggest non-technical skill gaps between industry and academia for the field of software engineering: *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, yielding 26 sub-themes grouped into six bigger themes: *stimulating continuous learning*, *stimulating creativity*, *creative techniques*, *addressing the gap in education*, *skill requirements in industry*, and the *industry selection process*. With this work, we hope to inspire educators to give the necessary attention to the uncovered skills, further mitigating the gap between the industry and the academic world.

CCS Concepts: • **Social and professional topics → Software engineering education**; **Computing profession**; *Computing industry*; • **Software and its engineering → Collaboration in software development**.

Additional Key Words and Phrases: software engineering education, creativity, professional skills, software developer, industry requirements

ACM Reference Format:
Wouter Groeneveld, Joost Vennekens, and Kris Aerts. 2021. Identifying Non-Technical Skill Gaps in Software Engineering Education: What Experts Expect But Teachers Don't Teach. *ACM Trans. Comput. Educ.* 1, 1, Article 1 (January 2021), 21 pages. https://doi.org/10.1145/3464431

1 INTRODUCTION

The software engineering (SE) field has long been known as a mainly technical field where knowledge about programming languages, intricate software patterns, and frameworks is needed to tackle problems and meet requirements. However, to be successful today as a software engineer, it no longer suffices to be technically proficient [15, 17]. Schumm and his team indicate the rising importance of teaching also other competences in addition to technical knowledge:

Authors' addresses: Wouter Groeneveld, KU Leuven, Leuven, Belgium, wouter.groeneveld@kuleuven.be; Joost Vennekens, KU Leuven, Leuven, Belgium, joost.vennekens@kuleuven.be; Kris Aerts, KU Leuven, Leuven, Belgium, kris.aerts@kuleuven.be.

1946-6226/2021/1-ART1 $15.00
https://doi.org/10.1145/3464431

> "*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.*" [36]

It has also been stated recently that industry regularly questions the work-readiness of graduates, and some companies are concerned less about their technical skills than about their *soft* skills needed for team based, customer focused business environments [38]. Educational institutions respond to these requirements by integrating these abilities into the curriculum. This was already proposed in 2007 as one of the main goals of new engineering technology programs [26].

However, as multiple recent studies have shown, many non-technical skill gaps still persist between academia and industry. For example, in 2014, Radermacher et al. investigated the skill gap between SE graduates and industry expectations by interviewing 23 employees of various software companies, concluding that efficient communication and problem solving skills are still major knowledge deficiencies [34]. They suggest that by exposing real-world software project problems to students, a part of the gap can be mitigated. In another study from 2014, Exter compared educational experiences with on-the-job needs of software designers [11]. The author also reports discrepancies in areas related to what she calls "practical" skills (as opposed to computing knowledge), such as communication, critical thinking, and problem solving. Again, interviewed participants suggest bringing large-scale real-world applications to the curriculum to balance the need for both theory and practice. These discrepancies were further validated by Exter et al. in 2018 [12], where a survey of almost 300 computing professionals indicated of the importance of a broad range of both technical and non-technical topics and skills.

According to Bailey, different non-technical skills are required depending on the job title and contents [5]. Nevertheless, the greatest common divisors among all reported non-technical skill requirements are again problem solving and various communicative skills such as listening and teamwork. The author concludes that:

> "*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.*"

In our own previous work [16–18], complementing the aforementioned research, we focused on identifying the non-technical skill gap in SE. First, we investigated which skills are currently being taught in SE education *according to the literature* (Study *SLR*). A systematic literature review (SLR) [18], analyzing 1962 publications from 16 countries, reveals self-reflection, conflict resolution, communication, and teamwork as the four skills that are taught most, while creativity, ethics, lifelong learning, and empathy dangle at the bottom. In total, 13 different skills were identified.

In a follow-up study, we investigated which skills are currently being taught in SE education, *according to learning outcomes of program syllabi* (Study *LO*). Learning outcomes (LO) of 278 computing program syllabi from 110 universities in 30 European countries [16] were carefully analyzed. 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. In total, 50 skills were identified.

Next, we conducted a Delphi study [17] to investigate which skills are required to be an exceptional software engineer *according to industry experts* working in enterprise software development companies (Study *DEL*). In this study, 36 experts identified and ranked 55 skills classified in four major areas: communicative skills (empathy, active listening, etc.), collaborative skills (sharing responsibility, learning from each other, etc.), problem solving skills (verifying assumptions, solution-oriented thinking, etc.), and personal skills (curiosity, being open to ideas, etc.).

In *SLR* and *LO*, we investigated which skills are currently offered to students, while in *DEL*, we identified the demands from the industry. However, the crucial question of how the two (mis)match remains. In this study, we aim to explore this issue by an analysis of the three existing studies, followed by a number of interviews. Thus, we ask:

- RQ1: *What are the most important non-technical skill gaps between academia and industry for the field of software engineering?*
- RQ2: *How are these skill gaps perceived by practitioners in the industry?*

The remainder of this paper is divided into the following sections. Section 2 describes background information related to work in the area, while Section 3 clarifies the utilized methodology. Next, in Section 4, we present the results of the gap analysis, followed by a discussion of the most important skills in Section 5. Limitations are described in Section 6. Section 7 concludes this work.

2 BACKGROUND

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 in the ACM SIGCSE Bulletin Journal of 1976 [29], 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 [8, 21]. 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.*" [29]

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 work [33], published in the 2013 ACM SIGCSE Proceedings, and that of Garousi et al. [15], 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 et al. continued to investigate the skill gap in 2014 by interviewing 23 practitioners in the field across the United Sates and Europe [34]. 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.

In 2012, Moreno et al. investigated the software engineering curriculum knowledge guidelines provided in *Software Engineering 2004* (SE 2004) [23] and *Graduate Software Engineering 2009* (GSwE 2009) [2], and the job profiles identified by Career Space [30]. 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 [3].

In 2015, Liedenberg et al. investigated the industry's perception of the relevance of software development education [25], 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 paper 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 mentions that taking ethics into consideration is rarely something a commercial company is after [35]. 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.

3 METHODOLOGY

3.1 Context: previous works

This research is part of a larger research program that we have developed in the past few years, with the ultimate goal to improve SE education. To investigate the non-technical skill gap, we further analyze the findings from our three previous studies, as visible in Figure 1.

In the first study [18], published in 2019 (*SLR*), we performed a systematic literature review by following a variant of Kitchenham's guidelines, which was specifically adapted for software engineering [41]. As data repositories, we used the ACM Digital Library and the IEEE Xplore libraries, to which the employed snowballing concept added further references of references. Our research question was: "*Which non-technical skills have been identified by educators as important to teach software engineering students?*". The 26 included papers mention 13 different skills in total, which are available in Table 4 in the appendix.

One of the biggest shortcomings of that study is the very nature of a SLR: undocumented successes or failures are not taken into consideration. To counter this, we conducted a follow-up study in 2020 (*LO*), investigating which skills are currently being taught in SE education, by manually curating a database of learning outcomes of computing program syllabi [16]. Our approach for this study was similar to the learning outcome analysis of Becker et al. [6] and selected the top 30% European universities based on the 2019 QS World University Rankings®. The research question used was: "*Which non-technical skills are taught in undergraduate computing courses, according to their learning outcomes?*" The 278 syllabi mentioned 50 different skills in total, which are available in Table 5 in the appendix. More information on the methodology is available in the course analysis paper.

The third previous work on which this paper is based, is a Delphi study presented at SIGCSE 2020 [17], in which we interviewed 36 SE experts from 11 different countries, affiliated with 21 internationally renowned companies and universities (*DEL*). The Delphi technique is an effective tool to reach a group consensus on a given topic, based on the opinion of experts [31]. The Delphi study consisted of four rounds: identification of skills (1), verification of results (2), selecting top skills (3), and ranking skills (4). Three panels were formed to cluster participants: technical experts (T), business experts (B), and academics (A). The research question used was: "*Which non-technical*

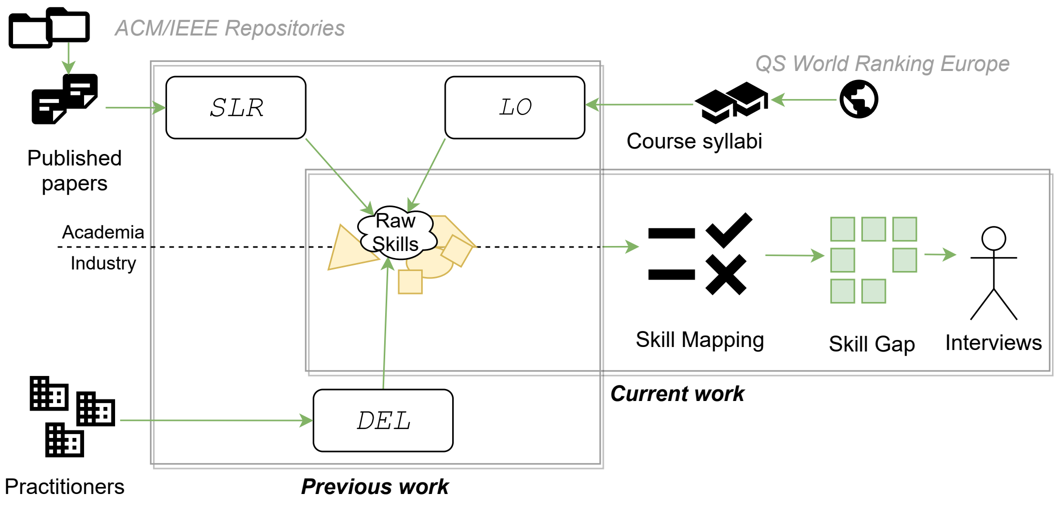

Fig. 1. The context of previous and current works.

skills are needed for a developer, to excel in the SE industry?" In total, 55 different skills were classified, of which 27 made it into round three, as visible in Table 3 in the appendix.

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 (as identified by studies *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.

3.2 The Skill mapping exercise

Different papers use different sets of SE topics: some use versions of the Software Engineering Body of Knowledge (SWEBOK) [7], while others use the "SE2014" curriculum guidelines outlined by ACM and IEEE [3]. Mapping SE terms has been done before by Garousi et al. in their SLR [15]. However, they fail to mention how exactly skills were mapped. In another study by Tariq et al., experts have been interviewed to audit and map generic skills development across a UK degree programme [40]. 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 (affiliates omitted for double-blind reviewing) 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 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 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.

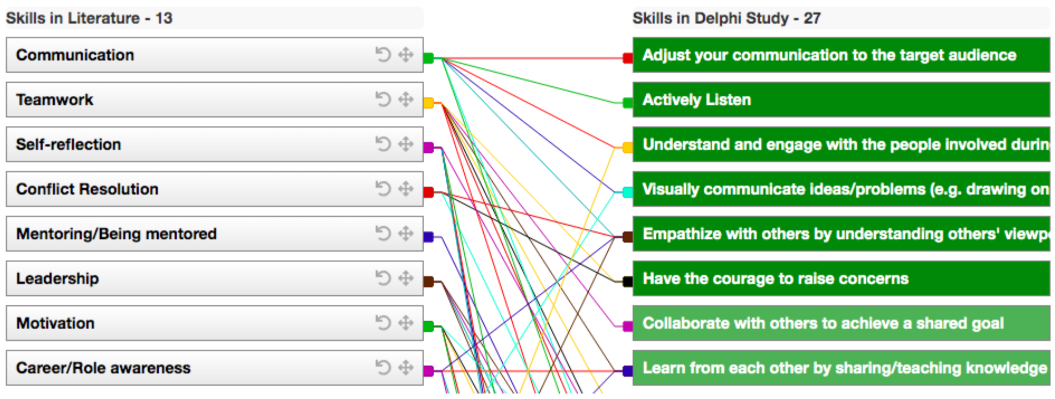
Skills in Literature - 13
Communication
Teamwork
Self-reflection
Conflict Resolution
Mentoring/Being mentored
Leadership
Motivation
Career/Role awareness
Skills in Delphi Study - 27
Adjust your communication to the target audience
Actively Listen
Understand and engage with the people involved durin
Visually communicate ideas/problems (e.g. drawing on
Empathize with others by understanding others' viewp
Have the courage to raise concerns
Collaborate with others to achieve a shared goal
Learn from each other by sharing/teaching knowledge

Fig. 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, '*communication*' has been linked to six different Delphi skills.

3.3 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. 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. Details about the Delphi methodology are available in [17]. Table 3 in the appendix contains a list of the included skills. 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.

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 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 count 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:

$$\frac{\sum_{P\in\{T,B\}} w_P(s)x_P(s)}{\sum_{P\in\{T,B\}} w_P(s)} \tag{1}$$

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 3.

The skills from the SLR and LO studies are mapped onto Delphi 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 Delphi skills was calculated. We will compare this average percentage to the Delphi score as defined above.

3.4 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., where industry veterans were approached based on their achievement of some degree of recognition as software engineering experts [24].

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 et al. [19] and Marshall et al. [27] to achieve this, as discussed in Section 4. 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 software engineering, as this is the same context as the Delphi study.

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 [13], Kontio et al. state that "*In the field of SE, it is very important for the interviewer to have extensive knowledge of the theme of the interview*" [22].

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 4). 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 guidelines for conducting semi-structured interviews in empirical SE research [20], 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 et al. [32] and McAlister et al. [28]. First, the audio recordings were transcribed by the first author. 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 [28] 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 first author using the final revision of the codebook.

4 GAP ANALYSIS RESULTS

From the Delphi study [17], the 27 skills that made it through round three were used as the target to map the 13 different skills found in the Systematic Literature Review [18] and the 50 skills found in the Learning Outcomes [16]. Eight experts helped us with this mapping exercise. After applying normalization steps explained in Section 3, the results were arranged in the skill comparison chart shown in Figure 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 4, which depicts the calculated delta values for each skill, categorized in the 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 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 1. The three largest "negative" skill gaps are *understanding and*

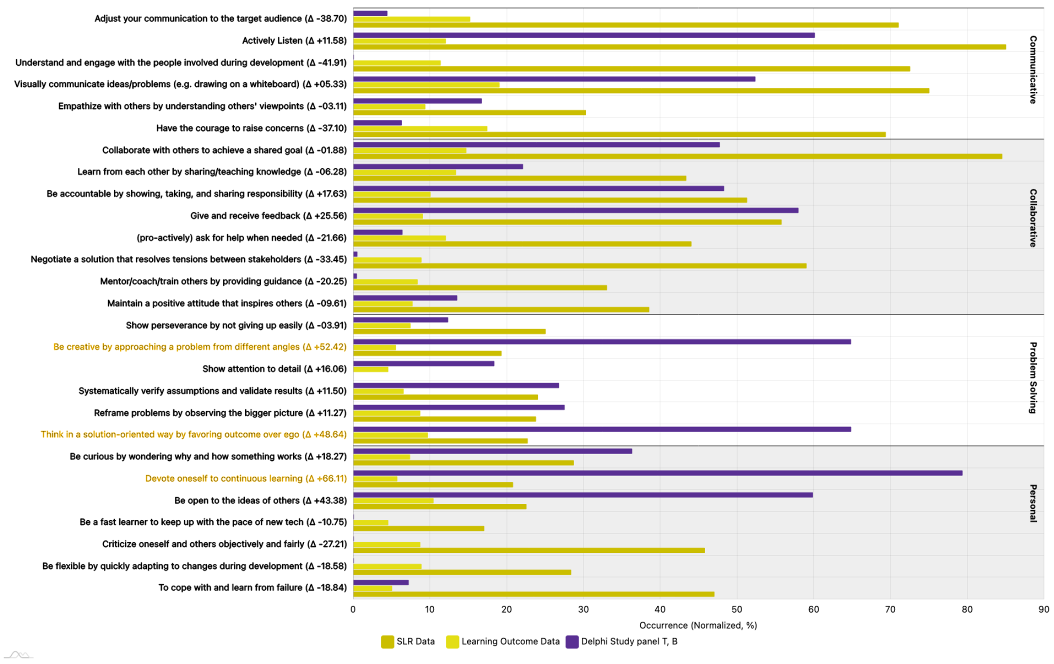

Fig. 3. The skill comparison chart, comparing data from the Delphi study (bar 1, in purple) to Learning Outcomes (bar 2) and SLR findings (bar 3). The bigger the difference between the yellow and purple bars, the bigger the perceived skill gap, as also indicated by the average delta values. An interactive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

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).

Table 1. The top three skill gaps with deltas for each study, normalized and expressed in percentages. Values between parentheses denote occurrences converted to percentages. The average delta (δ Avg.), as also visible in Figure 3, is the average of the SLR and Course percentages, compared to the Delphi occurrences.

Skill	Delphi occ.	δ SLR	δ Course	δ Avg.
Continuous learning	79.32	59.99 (19.33)	73.92 (5.40)	66.11
Creativity	64.81	49.81 (15.00)	54.01 (10.80)	52.42
Solution-oriented thinking	64.80	49.80 (15.00)	59.40 (5.40)	48.64

We conducted eight semi-structured interviews, which lasted on average 38 minutes. In total, 303 minutes were recorded, yielding 43604 transcribed words. Table 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. [19]. 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.

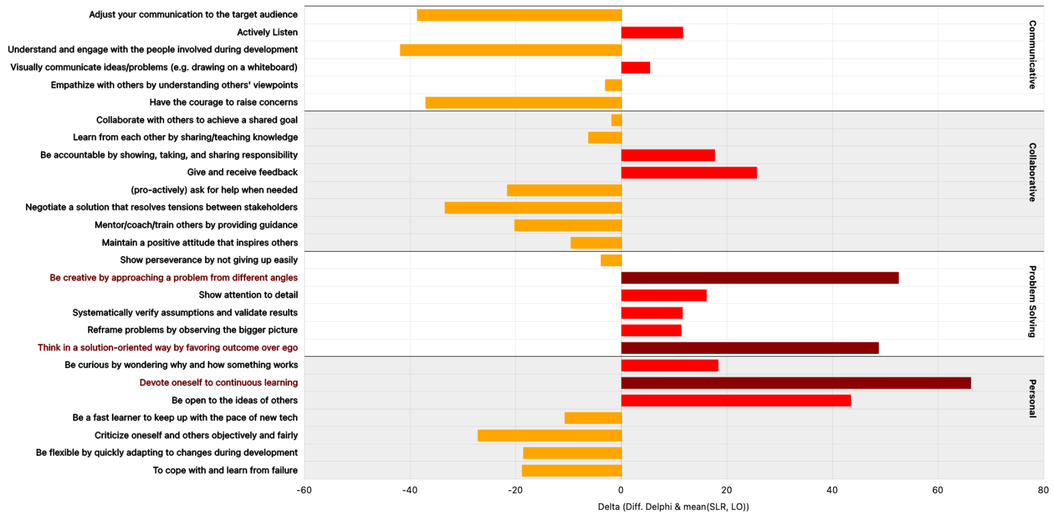

Fig. 4. An alternative view of Figure 3, depicting deltas for each skill group identified from the Delphi study. A positive delta ((dark) red) denotes a higher demand from industry compared to what is currently taught, while a negative delta (orange) denotes that the skill is included more in academia than in industry.

Table 2. Interview paradata, including saturation assessment data, using a base size of four interviews and run length of two. An information threshold of ⩽ 4% indicates data saturation.

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%	/

In the following section, we will briefly discuss each skill listed in Table 1, both in context of existing literature and in context of the conducted interviews.

5 INTERVIEW RESULTS

It is clear that all three skills from Table 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 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.

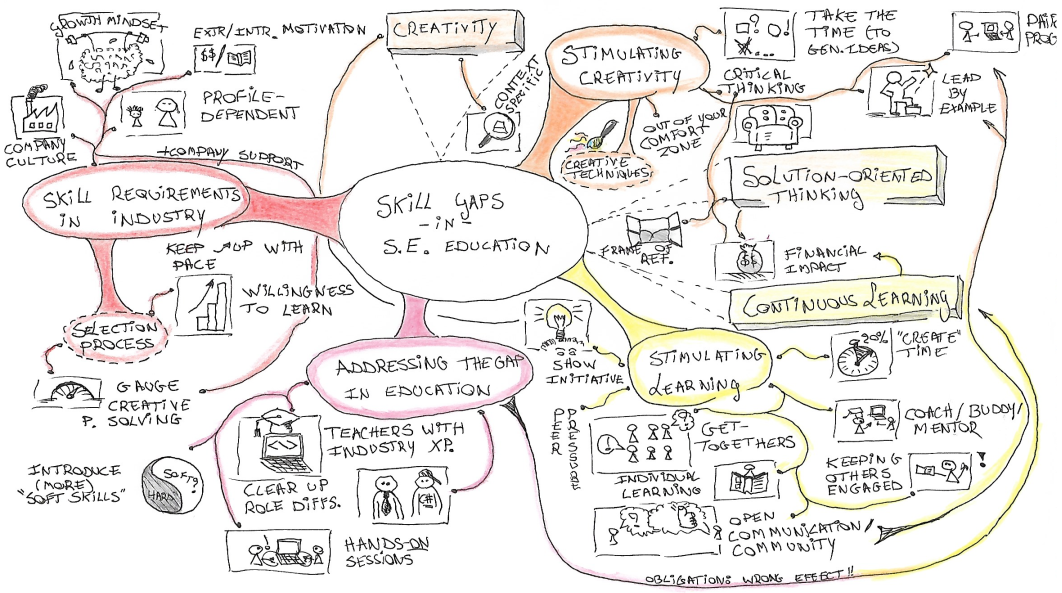

Fig. 5. A mind map that summarizes the themes we identified.

5.1 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 literature. Participants in a survey conducted by Exter et al. 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 [12]. 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[1]
6. Following internal Slack channels where interesting leads are posted and new technologies to pickup are discussed

One participant came up with a colorful analogy, originating from Sonmez' "Soft skills" [37]: 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 life-long 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[2], can help guide developers in the right direction.

5.2 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 [12, 14, 34]. Bailey mentions in her 2014 article 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

[1]https://dzone.com
[2]https://www.thoughtworks.com/radar/

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[3].

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 [9, 15, 24]. De la Barra et al. go as far as saying that "Agile Software Development is about knowledge management and creativity" [9]. 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 Componential Model of creativity [1].

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".

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

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."*

5.3 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 [34] and Exter et al.'s [12]. 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.

5.4 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 Liedenberg et al [25]. 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. mention in their 2018 publication 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*" [12]. 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 [39] 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") [4]. 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** in Learning Outcomes [16], as also noticed by participants.

5.5 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 Carol Dweck's "growth mindset" system [10]: 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 LIMITATIONS

Since we aggregated data from our previous studies to answer the research questions of this paper, 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 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. More information on the demographics of the Delphi participants is available in the paper [17].

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 paper: 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 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 3.4, 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 3.

7 CONCLUSION

In this paper, we aggregated and compared data about the non-technical skill gap in SE from previous studies, 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*. 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*.

Coming up with novel methods to teach and evaluate one of the three skills will be our next focus. As interviewees mention, stimulating continuous learning inside a limited academic context might not be very effective. Therefore, and since all three skills are closely related, our future work will involve zooming in on creativity and creative problem solving techniques.

ACKNOWLEDGEMENTS

We wish to thank all participants of the follow-up interviews and collaborators of the mapping exercise for their valuable contributions to this research. Thank you all very much!

APPENDIX

Table 3 contains the included skills from the Delphi study [17]. Table 4 contains the included skills from the SLR study [18]. Table 5 contains the included skills from the LO study [16]. More information is available in the respective papers.

Table 3. The 27 skills from the Delphi study per category that are utilized in this gap study. More details such as the number of occurrences and a ranking per panel are available in [17].

Communication skills (6)
Adjust your communication to the target audience
Actively listen
Understand and engage with the people involved during development
Visually communicate ideas/problems (e.g. drawing on a whiteboard)
Empathize with others by understanding others' viewpoints
Have the courage to raise concerns
Collaboration skills (8)
Collaborate with others to achieve a shared goal
Learn from each other by sharing/teaching knowledge
Be accountable by showing, taking and sharing responsibility
Give and receive feedback
(Pro-actively) ask for help when needed
Negotiate a solution that resolves tensions between stakeholders
Mentor/coach/train others by providing guidance
Maintain a positive attitude that inspires others
Problem-solving skills (6)
Show perseverance by not giving up easily
Be creative by approaching a problem from different angles
Show attention to detail
Systematically verify assumptions and validate results
Reframe problems by observing the bigger picture
Think in a solution-oriented way by favoring outcome over ego
Personal skills (7)
Be curious by wondering why and how something works
Devote oneself to continuous learning
Be open to the ideas of others
Be a fast learner to keep up with the pace of new technologies
Criticize oneself and others objectively and fairly
Be flexible by quickly adapting to changes during development
To cope with and learn from failure

Table 4. The 13 skills from the SLR study that are utilized in this gap study. More details such as number of appearances and identified teaching aspects are available in [18].

Skill
Communication
Teamwork
Self-reflection
Conflict resolution
Mentoring/Being mentored
Leadership
Motivation
Career/role awareness
Cultural intelligence/diversity
Creativity
Ethics
Lifelong learning
Empathy

Table 5. The 50 skills from the LO study that are utilized in this gap study. More details such as number of appearances and courses in which they appeared are available in [16].

Skill	
teamwork	ethics
written communication	presentation skills
oral communication	role awareness
critical thinking	set/keep timelines
self-reflection	adjust communication to audience
give/receive feedback	problem solving
conflict resolution	leadership
self-directed learning	accountability
effectively searching for data	creativity
risk assessment	interdisciplinary
coaching	work independently (within a group)
understand and engage with the people involved	negotiating
measure progress	group dynamics
entrepreneurial thinking	interact professionally
interpersonal skills	intercultural skills
manage priorities	react flexible to changes
measure self	show initiative
transversal competences	confidence in self
honour commitments	motivate self
innovative thinking	continuous improvement
stress resilience	draw on multiple sources for ideas
quality assessment	motivate others
improvisation	empathize with others
identify competencies	development of own values
expectation management	work systematically and methodically

REFERENCES

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

[2] Mark Ardis, Shawn Bohner, Dick Fairley, Massood Towhidnejad, and Art Pyster. 2009. Graduate software engineering 2009 (GSwE2009) curriculum guidelines for graduate degree programs in software engineering. *Integrated Software and Systems Engineering Curriculum (iSSEc) series* (2009).

[3] Mark Ardis, David Budgen, Gregory W Hislop, Jeff Offutt, Mark Sebern, and Willem Visser. 2015. SE 2014: Curriculum guidelines for undergraduate degree programs in software engineering. *Computer* 11 (2015), 106–109.

[4] DANIJEL Arsenovski. 2016. Swarm: beyond pair, beyond Scrum. *Experience report, Agile* (2016).

[5] Janet L Bailey. 2014. Non-technical skills for success in a technical world. *International Journal of Business and Social Science* 5, 4 (2014).

[6] Brett A Becker and Thomas Fitzpatrick. 2019. What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students?. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education.* 1011–1017.

[7] Pierre Bourque, Richard E Fairley, et al. 2014. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0.* IEEE Computer Society Press.

[8] Eric Brechner. 2003. 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.* 134–136.

[9] Claudio León de la Barra, Broderick Crawford, Ricardo Soto, Sanjay Misra, and Eric Monfroy. 2013. Agile software development: It is about knowledge management and creativity. In *International Conference on Computational Science and Its Applications.* Springer, 98–113.

[10] Carol S Dweck. 2008. *Mindset: The new psychology of success.* Random House Digital, Inc.

[11] Marisa Exter. 2014. Comparing educational experiences and on-the-job needs of educational software designers. In *Proceedings of the 45th ACM technical symposium on Computer science education.* 355–360.

[12] Marisa Exter, Secil Caskurlu, and Todd Fernandez. 2018. 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 (2018), 1–29.

[13] Edward F Fern and Edward E Fern. 2001. *Advanced focus group research.* Sage.

[14] Vahid Garousi, Gorkem Giray, and Eray Tuzun. 2019. Understanding the knowledge gaps of software engineers: an empirical analysis based on SWEBOK. *ACM Transactions on Computing Education (TOCE)* 20, 1 (2019), 1–33.

[15] Vahid Garousi, Gorkem Giray, Eray Tuzun, Cagatay Catal, and Michael Felderer. 2019. Closing the gap between software engineering education and industrial needs. *IEEE Software* 37, 2 (2019), 68–77.

[16] Wouter Groeneveld, Brett A Becker, and Joost Vennekens. 2020. Soft skills: What do computing program syllabi reveal about non-technical expectations of undergraduate students?. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education.* 287–293.

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

[18] Wouter Groeneveld, Joost Vennekens, and Kris Aerts. 2019. Software Engineering Education Beyond the Technical: A Systematic Literature Review. In *Proceedings of the 47th Annual SEFI Conference.* SEFI, 1607–1622.

[19] Greg Guest, Emily Namey, and Mario Chen. 2020. A simple method to assess and report thematic saturation in qualitative research. *PLoS One* 15, 5 (2020), e0232076.

[20] Siw Elisabeth Hove and Bente Anda. 2005. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05).* IEEE, 10–pp.

[21] Ivar Jacobson and Ed Seidewitz. 2014. A new software engineering. *Commun. ACM* 57, 12 (2014), 49–54.

[22] Jyrki Kontio, Laura Lehtola, and Johanna Bragge. 2004. 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.* IEEE, 271–280.

[23] Timothy C Lethbridge, Richard J LeBlanc Jr, Ann E Kelley Sobel, Thomas B Hilburn, and Jorge L Diaz-Herrera. 2006. SE2004: Recommendations for undergraduate software engineering curricula. *IEEE software* 23, 6 (2006), 19–25.

[24] Paul Luo Li, Andrew J Ko, and Jiamin Zhu. 2015. What makes a great software engineer?. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 700–710.

[25] Janet Liebenberg, Magda Huisman, and Elsa Mentz. 2015. Industry's perception of the relevance of software development education. *TD: The Journal for Transdisciplinary Research in Southern Africa* 11, 3 (2015), 260–284.

[26] David López, Fermín Sánchez, Josep-Llorenç Cruz, and Agustín Fernández. 2007. Developing non-technical skills in a technical course. In *2007 37th Annual Frontiers In Education Conference-Global Engineering: Knowledge Without Borders,*

Opportunities Without Passports. IEEE, F3B–5.

[27] Bryan Marshall, Peter Cardon, Amit Poddar, and Renee Fontenot. 2013. Does sample size matter in qualitative research?: A review of qualitative interviews in IS research. *Journal of computer information systems* 54, 1 (2013), 11–22.

[28] A McAlister, D Lee, KM Ehlert, RL Kajfez, CJ Faber, and MS Kennedy. 2017. Qualitative coding: An approach to assess inter-rater reliability. In *Columbus, Ohio https://peer asee org/28777: ASEE Annual Conference & Exposition*.

[29] Jan L Mize. 1976. Making an academic curriculum relevant to business requirements. *ACM SIGCSE Bulletin* 8, 2 (1976), 24–27.

[30] Ana M Moreno, Maria-Isabel Sanchez-Segura, Fuensanta Medina-Dominguez, and Laura Carvajal. 2012. Balancing software engineering education and industrial needs. *Journal of systems and software* 85, 7 (2012), 1607–1620.

[31] Chitu Okoli and Suzanne D Pawlowski. 2004. The Delphi method as a research tool: an example, design considerations and applications. *Information & management* 42, 1 (2004), 15–29.

[32] Anthony J Onwuegbuzie, Wendy B Dickinson, Nancy L Leech, and Annmarie G Zoran. 2009. A qualitative framework for collecting and analyzing data in focus group research. *International journal of qualitative methods* 8, 3 (2009), 1–21.

[33] Alex Radermacher and Gursimran Walia. 2013. Gaps between industry expectations and the abilities of graduates. In *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 525–530.

[34] Alex Radermacher, Gursimran Walia, and Dean Knudson. 2014. Investigating the skill gap between graduating students and industry expectations. In *Companion Proceedings of the 36th international conference on software engineering*. 291–300.

[35] Kevin Ryan. 2020. 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)*. IEEE, 103–106.

[36] Michael Schumm, Saskia Joseph, Irmgard Schroll-Decker, Michael Niemetz, and Jürgen Mottok. 2012. 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)*. IEEE, 1–5.

[37] John Z Sonmez. 2016. *Soft skills: The Software Developer's Life Manual.* Manning Publications,.

[38] Matt Stevens and Richard Norman. 2016. Industry Expectations of Soft Skills in IT Graduates: A Regional Survey. In *Proceedings of the Australasian Computer Science Week Multiconference* (Canberra, Australia) *(ACSW '16)*. ACM, New York, NY, USA, Article 13, 9 pages. https://doi.org/10.1145/2843043.2843068

[39] Jeff Sutherland and Hugo Heitz. [n.d.]. Scrum and Lean: How a Lean Scrum Can Improve Your Performance. ([n. d.]).

[40] Vicki N Tariq, Eileen M Scott, A Clive Cochrane, Maria Lee, and Linda Ryles. 2004. Auditing and mapping key skills within university curricula. *Quality Assurance in Education* (2004).

[41] Claes Wohlin. 2014. 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*. ACM, 38.