IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 3, 2019, accepted July 30, 2019, date of publication August 21, 2019, date of current version September 4, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2936660

Perspectives on the Gap Between the Software Industry and the Software Engineering Education

DAMLA OGUZ[1] AND KAYA OGUZ[2], (Member, IEEE)
[1]Department of Software Engineering, Yasar University, 35100 Izmir, Turkey
[2]Department of Computer Engineering, Izmir University of Economics, 35330 Izmir, Turkey

Corresponding author: Kaya Oguz (kaya.oguz@ieu.edu.tr)

ABSTRACT The gap between the software industry and software engineering education was first mentioned three decades ago, in 1989. Since then, its existence has been regularly reported on and solutions to close it have been proposed. However, after thirty years this gap resists all efforts for closure. In this study we assert that the gap between industry and academia exists for several reasons that are related and intertwined. To take a broader look at the problem from the perspective of all related entities, we (i) provide a detailed overview of the profession and identify the entities, (ii) extract the causes that stem from these entities and discuss what each entity should do, (iii) report and analyze the results of a questionnaire that has been conducted with students and recent graduates, (iv) emphasize the highlights of the interviews conducted with students, recent graduates and academics, (v) and compile a list of skills that are sought by the industry by analyzing the software engineering job advertisements. We further contribute to finding solutions by considering all entities involved, which provides an opportunity to access all of them, so that each can find out what they can do to acknowledge and narrow the gap. Our study concludes that the gap requires constant attention and hard work for all of the entities involved, and therefore all should be on the lookout for new technologies, learn to embrace the changes and adapt to them, so that the gap is kept at a minimum.

INDEX TERMS Software engineering education, education gap, engineering curriculum.

I. INTRODUCTION

Software engineering is a challenging profession in many aspects. The foremost challenge stems from the nature of the software itself. Compared to other engineering disciplines, the software product is not tangible and does not obey any physical laws which makes it rely on good practice rather than a fundamental theory [1]. This abstract nature also affects its design, since as the size of software increases, so does its design complexity, which puts a strain on the shoulders of software engineers [2]. Another challenge arises from its manufacturing process. On top of the complexity of design and implementation, human factors such as communication and teamwork, come into play during the software development process [3].

Students who take up the challenge of becoming software engineers complete a set of courses, develop course projects, complete their internship, and demonstrate their capabilities with a capstone project [4]. However soon after graduation, they notice that real life projects are of a different breed from the ones they have handled during their education [5]. This situation creates the famous gap between industry and software engineering education. In a very broad sense, we can define the gap as the differences between software engineering education and the software industry that are reflected in the associated entities. The industry expects competent graduates [6], while the students emphasize that their initial experiences in industry have been rather different than their education [5].

The gap is acknowledged in other engineering disciplines, too. Alboaouh emphasizes the two to three years of time required for the recent graduates to gain experience in the engineering profession [7], while Goold finds out that the engineering students are trained more in theory than in practice [8]. Software engineering, in this regard, has its own challenges that creates its own unique case. As mentioned, the foremost challenge is handling the intangible and highly malleable nature of the software which makes it rely on good practice rather than laws of nature [1]. Even though the students grasp the theory and apply it to problems that

The associate editor coordinating the review of this manuscript and approving it for publication was Haider Abbas.

 This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

are tailored for a course, more experience is needed for handling real life projects with ambiguous specifications. On the other hand, as a profession, software engineering is not only a young but also a rapidly changing industry with a vast range of applications. The advances in associated technologies create new and more capable hardware devices, which in turn create new platforms, programming languages, libraries, development environments, novel software processes, and specialized jobs while the student is still working towards a typical four-year degree in software engineering. Perhaps the most relevant examples would be the introduction of smart phones a decade ago which became devices more personal than personal computers, and the more recent introduction of Internet of Things (IoT) which redefined the meaning of a host in the Internet as not only a computer but any device which has networking capabilities. In this regard, developing a mobile application is considerably different from developing an IoT system that is made up of several systems and devices, since the platforms, target users, and objectives are significantly different in both cases, while the profession is still the same. It is not surprising, then, to have a gap between the software industry and the world of academia considering the intangible nature of the software and the rapid changing state of the profession.

The gap is mentioned as early as 1989, in the opening sentences of Ford and Gibbs' recommendations on a master of software engineering curriculum, at a time when an undergraduate degree in software engineering was not offered because the discipline was considered immature [9]. Almost a decade later, Beckman *et al.* proposed a model for collaboration between industry and academia to close the gap between them [10]. The model suggests the formation of an advisory board of industrial representatives to identify common objectives and goals. By doing so, the activities can be planned to meet these objectives. The progress should be regularly measured and feedback should be given. Around the same time, Shaw proposed a road map for software engineering education in which she has emphasized keeping education current with the rapid changes in the field [11]. Her foresightful suggestions include the consideration of open source development for keeping up with practice and encouragement of continuous self-learning. Also, Parnas has proposed a curriculum for software engineering education that is radically different than computer science education in its curriculum philosophy, course style and content, and topic coverage [12]. Another decade later, Begel and Simon have reported the results of their ethnographic study about the struggles of the new software developers [13]. They found out that while new developers were capable of design and implementation, their communication, teamwork and orientation skills were lacking. Consequently, the authors suggested a reform in the curriculum of computer science education. Today, the gap is still a relevant topic of interest as the recent study by Craig *et al.* reports the analyses of interviews of twenty early career developers [5]. Apparently the gap resists closure after three decades of its existence in spite of several analysis, reports, suggestions and applications.

As academics with industrial experience, it is impossible not to notice the fact that most of the students will struggle in an industrial environment when their exams and course projects are evaluated. Although formal feedback of students is received regularly via the mechanisms that are inherent in the system, casual and informal conversation can provide more insight because the matter can be tracked by asking follow up questions immediately in the vicinity of the session. Such conversations with students, as well as fellow academics, lead to the conclusion that the gap still exists, and will continue to exist because of several causes.

In this study we assert that the gap between industry and academia exists because

- there is no single source of the problem,
- the software engineering profession is able to quickly react to new platforms and trends that require the acquisition of new skills while academia is not,
- academia is not quick enough to incorporate the changes in the profession into its curriculum,
- it is challenging to create realistic experiences in engineering education,
- lack of realistic experiences make it more challenging for the students to acquire soft skills that are needed for the collaboration to develop large-scale software projects.

We contribute to the literature about our assertion of the persistent gap by offering the following to narrow and close it:

- analyzing the gap from various different perspectives that involves all related entities,
- using questionnaires and conducting interviews with students, recent graduates and academics to get comments and views on the problem,
- compiling a list of skills that are sought by the software companies, composed by careful inspection of fifty software engineering position advertisements,
- presenting our own conclusions and discussing proposed methods that will help students and software engineers to bridge the gap.

The remainder of this paper is structured as follows. Section II presents an overview of the software engineering profession to identify its major and minor entities on different dimensions. In Section III, various perspectives on the gap are discussed in detail. Section IV includes the feedback from students, recent graduates and academics. This section also includes a survey of skills sought in software engineering by analyzing current software engineering job advertisements, and includes remarks of industry practitioners from the existing studies in the literature. In Section V, we give a summary of what each entity should do to identify, acknowledge and narrow the gap. The paper concludes with comments on the current status and the future of the gap in Section VI.

II. AN OVERVIEW OF SOFTWARE ENGINEERING

We asserted that the gap between the industry and the education exists because there is no single source of the problem. It is then necessary to take a step back and look at the field from a higher point of view. In Figure 1 we identify several entities that form and shape the profession along with their connections to either the academia or the industry. The arrows show binary relations, while the diamonds are used to represent multiple relations. The distinction between the academia and the industry are not always clear cut for some entities and practices. However, it is essential to point them out because the gap is between them, by definition.

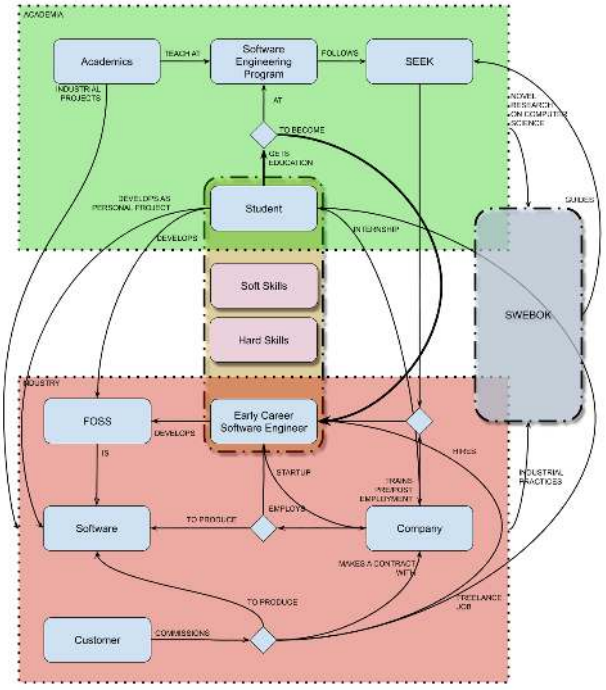

FIGURE 1. A view of the entities that form and shape software engineering. Both the industry and the academia contribute to the body of knowledge of the profession, while the hard and soft skills are the abilities of the students and software engineers. The arrows show binary relations, while the diamonds are used to represent multiple relations.

Figure 1 contains the entities that are in the academia and the industry, as well as two additional dimensions that are orthogonal to each other and to the dimension of the entities. Software Engineering Body of Knowledge (SWEBOK) is composed of fifteen Knowledge Areas (KA) along with seven related disciplines [14]. It is bound neither to the academia nor to the industry, since it encompasses and affects both of them while also being affected by both. It includes everything about the profession which makes it, in a sense, the profession itself. The skills dimension represent the abilities of students and software engineers. They are acquired by both education and experience, depending on their type and the context. The figure includes their categorical view; the hard skills are also called technical skills, such as programming, while the soft skills are also called social skills, such as written and oral communication. They are essential because a student becomes a software engineer by getting education via a software engineering program in order to gain and improve their relevant and related skills. They are also essential in the sense that they are used to evaluate software engineers when they are considered for a position in the industry.

As can be seen in the Figure 1, every entity is related to others without a clear hierarchical structure. This creates a challenge for those who want to study them in isolation, while they should be studied as a whole. We define students as candidates who are getting an education in a software engineering program that follows the guidelines in Software Engineering Education Knowledge (SEEK) in order to become software engineers. The academic institutions and accreditation agencies for software engineering programs follow SEEK, which in turn is recommended by SWEBOK [4]. Students are expected to acquire both hard and soft skills to make them competent in the profession. They do so by completing their education, developing personal projects, contributing to free and open source software (FOSS), and acquiring freelance jobs from customers, as the arrows represent in the figure.

Students become software engineers when they are awarded a degree from the software engineering program. The figure intentionally refers to them as early career software engineers (ECSE) because they will, hopefully, bridge the gap when they become senior engineers. The same sources of experience are also valid for the ECSE, however, there are cases where companies choose to apply pre-employment or post-employment training to software engineers to improve the skills of their employees [15], often using a curriculum based on SWEBOK [16], [17].

The academia has two major contributions to the profession; the first one is to train and educate students to become software engineers. The second contribution is provided by the academics, who do novel research in the foundations of computer science, mathematics and engineering. The academics are the executors of the teaching process while also being voluntarily engaged in industrial projects to keep track of the current practices in the field.

The software engineering industry is where the software production takes place. It mainly contributes to the profession by producing its own practices. Its most important entity is the software itself which requires a development process that must overcome the challenges in realizing an abstract product. The customer can commission the development of software via a company that employs software engineers, or by hiring software engineers as part of its own infrastructure, or by students who are looking for freelance jobs.

FOSS enables collaborators from all around the world to develop software that is publicly available [18]. Naturally, it created novel practices that proved their worth by producing high quality software. For the students, software engineers, companies, customers, and all who are involved in software, FOSS proves to be a useful resource not just by providing free and open software, but also by providing an opportunity to get experience in the development processes with minimal costs.

TABLE 1. The table lists the causes of the gap from various perspectives. Each line lists the source of the cause, the cause in brief, the solutions, and the parties that involve the solution. 'I' denotes the industry, 'A' denotes the academia, 'S' denotes SSE, and N/A denotes not available. If the solution has a reference, then it is an existing solution, otherwise the solution is a proposition by the authors.

Source	The Cause	Solutions	I	A	S
N/A	The software industry expands to new areas.	No solution, unless we want to stop improving			
I	The feedback from industry to academia is missing.	Industry should be involved in the education of professionals via a board of industrial representatives [10]	✓	✓	
		Publish and share the experienced gained in the new platforms that have not reached the academy yet.	✓		
I, A	Software engineering education is not agile enough to incorporate the new practices that emerge in the industry	Holistic approach to curriculum [19], [20]		✓	
		Problem-based learning [21]–[23]		✓	
		Studio-based learning [24]–[27]		✓	
A	Courses are isolated, but actually related, which prevents the students to see the connection between them	Course projects must include the topics in the previous prerequisite courses		✓	
		Gradual updates to the curriculum [28]		✓	
I, A	Academics are out of touch with the industry	Summer internship at the industry [5]	✓	✓	
		Get involved in industrial projects	✓	✓	
S	SSE may widen the gap.	Include self-regulated learning strategies into the courses [29]		✓	✓
		Motivation to contribute to projects and realization of the benefits in the long run			✓
A	Soft skills are required for software engineering practice	Course projects that incorporate teamwork. [30]		✓	
		Call to promote soft skills in software engineering [31]		✓	
		Promote software engineering programs as both technical and sociotechnical to prospective students		✓	
		The technical and introvert male stereotype should be remedied, the profession should be promoted to a more diverse set of prospective students		✓	
		An entire course on the human aspects of software engineering [31]		✓	
		SWEBOS - Software Engineering Body of Skills [32]	✓	✓	
		Academics should involved in student projects to demonstrate team skills		✓	
		Use pre-employment or post-employment training [15], [16]	✓		✓
A	Course projects don't have realism	Incorporate FOSS [33]–[36]	✓	✓	
		Introduce real customers [37]	✓	✓	
		Use academics or senior students as mentors [37]		✓	✓
		Use a codebase that is developed by students for teaching maintenance [38]		✓	✓
		Use problem based learning to have technical and soft skills, as well as software quality [21]–[23], [39]		✓	
		Academics should be involved in projects that are interesting enough to attract students using recent and up-to-date technologies		✓	

The overview of the profession shows us that in such a connected profession, it is not possible to close the gap by considering only some of the entities. A change in one of the entities create a ripple that reach other entities that require a revision to software processes, skill sets of the software engineers, new topics in academia, and finally an update to the profession that will be reflected in SWEBOK. These ripples are common in the ever vibrant field of software engineering as exemplified earlier.

III. PERSPECTIVES ON THE GAP

The gap is defined as the differences between software engineering education and the software industry that is reflected in the associated entities. We have defined these entities in Section II as the ones that make up the industry and the academia. We have mentioned that SWEBOK encompasses and affects all the entities, and is affected by all of them. Also, we have defined the skills as the abilities of both students and software engineers.

To analyze the gap from all perspectives, we have categorized these entities into three major groups: SSE, industry and academia. The first group is labeled as SSE, since it is made up of the student and software engineer entities. They are placed in the same group because we perceive them as the same entity being represented at different points in time. Additionally, they embody the skills dimension within themselves. The other two groups are the remaining entities in industry and academia, respectively, and they are labeled as such.

The first part of the perspectives include the self analysis of each group. The second part of the perspectives discusses the causes of the gap, and how they involve the related groups. An overview of these causes, their solutions and references are given in Table 1.

A. SELF ANALYSIS OF GROUPS

The first set of perspectives considers each of these groups on their own. We have identified the causes that stem from the

elements in these groups, how they contribute to the gap, and how they can be mitigated, or if possible, removed.

1) THE INDUSTRY

How does the software industry contribute to the gap? The software industry explores and conquers new horizons that appear frequently thanks to the technological advances in associated fields. In Section I two of these new platforms were discussed, the mobile platform and the IoT. Another one is the current state of the JavaScript frameworks [40], their problems [41], [42], and how to pick one [43].

Such technologies can create a paradigm shift which challenges all aspects of software. It is self evident that programming languages, libraries, data types, data storage, software distribution, software processes and anything related are subject to change. The new paradigm could take over the industrial practices while the student is still studying at a software engineering program, which typically takes four years to complete. The software industry widens the gap between itself and the academia, but it has no choice, because it has to adopt these new technologies to survive economically. Its adoption is also essential because it expands our knowledge and experience in this relatively young discipline.

Alboaouh asserts that while both the industry and academia contribute to the advances in research, the feedback from the industry to the educational bodies is missing [7]. Since the recent graduates are missing practical knowledge, the industry favors graduates with high GPA to reduce their training time. He also asserts that the industry is not enthusiastic about reforming the education process because training new engineers is much cheaper. While his comments are about the engineering disciplines in general, we believe that the advances, methodologies, and techniques that are developed in the software industry emerge too often and they need time to mature until they can be included in the curriculum. However, it should also be stated that the industry should be involved with the education of professionals through institutions and organizations such as IEEE. As Beckman *et al.* have proposed, a board of industrial representatives can help academia to define common objectives and goals [10].

2) THE ACADEMIA

The academia has two major entities, the software engineering program, which is typically the undergraduate programs in the universities, and the academics who both teach and contribute to the body of knowledge with novel research.

The academia is well aware of the expectations of the software industry, however, it is not quick enough to make a course correction at the same speed. It cannot afford trial and error in education, therefore, it has to take its steps more carefully and slowly. Nevertheless, there are cases where the academia makes significant updates to its curriculum to catch up and to educate and train software engineers with skills that are relevant in the industry.

Software engineering programs follow SEEK, the guideline recommended by the SWEBOK [4]. SEEK contains guidelines on the design and delivery of a curriculum. We would like to emphasize the ones that we find relevant to narrowing the gap before we discuss the existing and new solutions. The curriculum guidelines put emphasis on both the basic principles of software engineering and the usage of current tools, so that the education will not be out of date. Another guideline states that the curriculum should have a significant real life basis by incorporating case studies, project-based activities, a capstone project, practical exercises, and student work experience. These activities help the students to gain experience by handling larger projects as a group. The student work experience, which is commonly executed as an internship, allows the student to experience real life projects with real stakeholders and users. The guidelines also state that the curriculum should consider various approaches in teaching and learning, such as problem-based learning (PBL), just-in-time learning, learning by failure, and technology-enhanced learning. Among them, problem-based learning is a common approach in many fields and disciplines such as computer engineering when integrated with content-based learning in [44], and software engineering education is no exception.

In PBL, the students learn during the process of solving a problem that is put forward either by the lecturer or the stakeholders involved, such as real customers. It has been successfully applied to software engineering since several studies report that they have implemented it [21]–[23], while some point out the limitations, such as the difficulty of starting a project before learning about the topics in the course [45], or the cost of setting up a software development system and its environment [46].

Although not listed in the SEEK guidelines, there are other approaches that can help the gap to narrow. One of them is the studio-based learning (SBL) where a group of students work together to solve a problem in a defined space, such as a laboratory or a studio, as the lecturer mentors them and gives lectures only when necessary. While the idea is borrowed from arts and design departments, software engineering can make use of it since it is also a creative activity. Nurkkala and Brandle report that in their case the admission to the SBL track requires the permission of the instructor who assesses the prospective students by interviewing them [24]. The students who can perform well without constant oversight are more likely to be admitted. The program uses Scrum in the in-class lab sessions and use team software development for projects. Other studies have also reported successful implementations [25]–[27].

Another example is the holistic approach by Ellis *et al.* which discusses the usage and incorporation of FOSS into the software engineering curriculum [47]. They report that the students have gained more experience in software engineering knowledge and experience in areas such as gathering requirements and understanding software quality.

While a major update to the curriculum using a holistic approach is lacking in software engineering departments, there are studies in electrical and computer

engineering departments. Rashid and Tasadduq propose a holistic approach to computer engineering curricula by introducing the component level courses, and then merging them into system-level courses using Y-chart methodology [20]. The authors emphasize the forward and backward references of these courses which helps the students to see the bigger picture by either forward referencing an important topic in a future course, or by backward referencing an acquired knowledge to build on top of it. Maciejewski *et al.* make a bold move with the support of a five-year grant and approach the electrical and computer engineering education in a holistic fashion by stressing knowledge integration, collaboration among faculty and students, and making use of thematic threads [19]. The authors discuss their partial results which report that the students, while hesitant at first, have enjoyed the hands-on and collaborative approach. Although these studies are meant for computer engineering education, we believe that software engineering has close ties to computer engineering and a software engineering program may follow the suit.

Luukkainen *et al.* have worked for three years and updated their curricula gradually to fine tune it in order to improve the student's readiness for the practices in the industry [28]. The resulting curriculum has courses that are linked strongly together, building on a theme of ideas across them. The authors state that the success depends largely on a specific course called Extreme Apprenticeship which improves the programming skills of students under the guidance of more experienced programmers.

Curriculum design and delivery is a major undertaking which is under the regulation of a set of bodies, such as accreditation agencies, laws on higher level education, and the regulations of the universities. Therefore it is not possible to make significant alterations immediately. However, the courses, their learning outcomes, and their connections can be improved gradually.

An important cause of the gap is that the software engineering education is made up of courses that are connected and built on top of each other, but the students fail to see their connection. It is also common for the academics to focus on the topics related to the course and overlook other topics. Students get proficient in each topic, but they never develop complete applications where every topic is used at the same time. If the course is a data structures course, the project evaluation may overlook the graphical user interface, if the course is about human computer interaction, then the evaluation may overlook the data representation, or the object oriented principles. We propose a solution that requires the course project to include topics in the previous prerequisite courses. This solution may also require an update to the prerequisite course list of some courses in the existing curricula. It also requires each project to be more comprehensive which will create a gradual increase in difficulty and complexity, so that the transition from programming to software engineering can be handled more gracefully.

Craig *et al.* report that they have found some of the academics to be out of touch with industry, and list it as one of the causes of the gap [5]. They propose, as a solution, summer internships at the industry so that the academics can catch up. We also would like to add that the academics can get involved in industrial projects to get back to the reality of the industry.

3) STUDENTS AND SOFTWARE ENGINEERS (SSE)

Students become software engineers after their graduation, and they are the ones who feel the gap the most. They become aware of the gap either when they are employed at a company, or when they first experience the industry environment as a student, probably during their internship. They are the most unfortunate of the three major groups because they have less experience than the entities in both the industry and the academia, but they are expected to bridge the gap as soon as possible.

The gap is reflected directly in the skill sets of SSE. They are the realization or the embodiment of the gap. From this perspective, they can affect the gap in both directions; they can narrow and close it, or they can make it wider.

The evaluation of the industry shows us that there will be new technologies even after the graduation, so the student should be always vigilant and ready to learn them. The evaluation of the academia shows us that students who are interested in the profession are more likely to succeed. Falkner *et al.* have reported the success of their students who were able to master the self-regulated learning skill, which lets the students set their goals, and organize their resources to achieve their goals [29]. The guidelines in SEEK also emphasize the importance of continued learning and self learning [4].

These findings point out that the individual SSE has to be involved in several software projects to improve both their hard and soft skills, while learning how to learn on their own, and to find out ways to overcome the problems and difficulties when they face them. We believe that, just as the industry has to fill the software void created by the new associated technologies, SSE has to be ready to embrace new technologies, quickly adapt to them, and update themselves with new skills.

While the following suggestions can apply to recent graduates as well, we believe they are more appropriate for the students. On top of the list, the students should be motivated to contribute to the projects they are involved in. While this largely depends on the project and the personal preference of the student, they should consider the benefits in the long run where each project brings in more experience. Involvement in projects also increase the time invested in programming which is a good sign of gaining experience.

B. CAUSES OF THE GAP

We have identified several causes of the gap which originate in one or more of the major groups and collected them under related categories that follow.

1) THE SOFT SKILLS

Although there are many studies that stress the importance of soft skills in software engineering [3], [48]–[53], they are also the most popular cause of the gap since early career software engineers report that they do not feel they are prepared for communication and teamwork, where the soft skills will be the most useful [5].

The common solution to close the gap for this cause is to have projects that incorporate teamwork [30]. Capretz emphasizes that software engineering has human factors in its processes [54], and with Ahmed they make a call to promote soft skills in software engineering [31]. We extend their proposition and make a call to software engineering programs to advertise software engineering not only as technical, but also a sociotechnical practice. Software engineering is most related to computer science and computer programming which has stereotype individuals as being very good at technical skills but awkward in social situations. Studies on stereotypes in engineering in general is surprisingly low. Bailey *et al.* stress that engineering and computer science is male dominated [55], while other studies focus on the introvert stereotype of engineers [56], [57]. Studies show that people underperform when they are under situations that cause a stereotype threat and there are ways to reduce it [58]. Therefore, we think that the software engineering programs should appeal to a wider range of prospective students if it is promoted as a more sociotechnical practice, turning the profession into a more social one and overcoming the perception of introvert male stereotype.

Another proposed solution is also made by Capretz, who suggests having an entire course on the human aspects of software engineering [54]. Sedelmaier and Landes propose a body of skills, SWEBOS, akin to SWEBOK, which describes the required skills for software engineering [32].

The requirement for soft skills are widely acknowledged but very rarely emphasized in software engineering programs. We hypothesize that the academics acquire these skills on their own, either by their industrial experience, or because of their research environment which is collaborative by nature. Therefore, they expect the students to acquire these skills when an environment for collaboration, such as a team course project is provided for them. The academics should also be involved, either directly or indirectly via teaching assistants or more experience programmers to demonstrate how a team should operate and help the students gain team experience and sharpen their social skills.

2) REALISM IN COURSE PROJECTS

The software engineering programs provide practical experience through laboratory work, course projects, and a final capstone project in which the students demonstrate all their skills, as recommended in SEEK [4]. These projects are also expected to provide the students with the opportunity to gain and improve their soft skills, as discussed in the previous cause.

Early career software engineers and students report that they do not find the course projects as realistic as the real industrial projects [5], [37]. These projects are usually called toy projects, because they are smaller in scope compared to the real projects, their requirements are well defined, and the academics act as both the customer and the mentor. Simpson and Storer report that the students find the lecturer acting as an uncertain customer to be malevolent and implausible [37]. They also state that the teams are made up of all novice engineers, which strips them of the benefits of good leadership that might be provided by experienced engineers. However, mentoring all teams does not scale well, the academics cannot handle large classes because of time constraints.

Realism in course projects is the most studied cause of the gap because the skills required to handle real life projects are expected to improve with a hands-on approach, and failing to provide realism deprives the students of valuable experience. The most common solution to this cause is to incorporate FOSS projects [33]–[36]. This approach is successful in most cases, but it does not remove the problem completely. Studies report that it is not that easy to find appropriate projects, students may not be able to get involved in FOSS communities [35], and the features that will be implemented for a FOSS project may be considered as toy features by the student [34].

Simpson and Storer state that they have introduced real customers for the course project, and have started a leadership course where senior students acted as mentors for these industrial projects [37]. They report having convincing results for the students, since they had a realistic experience, the mentor students got their chance to pass on their experiences, and the customers were more than happy to get quality software at very low costs.

Other studies have reported that software maintenance and quality should be emphasized [38], [39]. Szabo states that they use a code base for software maintenance that is developed by students in previous semesters. This brings down the quality of the code to a level that is appropriate for software maintenance, which makes it realistic.

The realism of the course projects are illusionary because they are tailored for the course and the students will be finished with those projects as soon as the course is over. It might be possible for the lecturers to gain their attention by starting interesting projects that use recent and up-to-date technologies. While these would not be a course project, they would be a project developed within the lecturers supervision. Such projects would benefit both the lecturer and the students. The lecturer would stay current with recent developments in the industry, the students would have the chance to volunteer for a real project that is supervised by an experienced engineer. While the project could be developed in an open source fashion, it does not necessarily have to be that way. We believe it could be useful as long as the students could be involved, have regular meetings, design sessions, and have the ability and the opportunity to contribute to the source code.

3) INDUSTRIAL TRAINING

Since the gap exists, the software companies have to find a way to deal with the reality that new software engineering graduates have gaps in their skill sets. In order to overcome this problem and improve the skills of their employees, they consider either pre-employment or post-employment training.

The pre-employment training is detailed in the study of Tuzun *et al.* where they identify key knowledge areas which their candidate employees are lacking in skill [16]. They create a ''summer school'' and train the graduates in these knowledge areas using the curricula presented at the universities, whose curriculum are guided by SEEK. They hired several of the candidates according to their performance. The candidates reported back that they found the summer school useful, even if they are not hired, because they have improved their skills.

The post-employment training was done at least once at Microsoft [15]. Brechner reports that Microsoft stopped all development on Windows for months to train over eight thousand employees on writing secure code.

With several new software engineers being trained every year, we believe that companies will require more scrutinized interviews when hiring, or will make use of trial periods before they will hire new talents.

IV. INTERVIEWS, QUESTIONNAIRES, SURVEYS

We have gathered insights, comments, and data from the major bodies we have defined earlier using various approaches to compare them to the existing research discussed in Section III.

The first set of data comes from an online questionnaire that is sent to several students and recent graduates of at most two years from three different universities using personal contacts and word of mouth. For the second set of data, we have used our personal contacts to interview students and recent graduates either face-to-face or online. For the set of data that involves the views of academics, we again used face-to-face and online interviews. The final set of data comes from the current job advertisement on software engineering. We have compiled a list of skills that are sought for software engineering positions in these advertisements.

A. THE QUESTIONNAIRE

The questionnaire has reached 78 students and recent graduates. It is made up of ten questions that are either multiple choice, multiple answers, or open ended. The questions are thematically about the self evaluation of their soft and hard skills.

The questions and the replies provided are detailed in Tables 3 and 4.

The first question, **"What is your year of study in the software engineering education program?"**, helps us to differentiate current students and graduates. The answer is given in a multiple choice list, years ''1'' to ''4,'' with ''4+,'' and ''graduate'' being the final choices. The years corresponds to freshman, sophomore, junior and senior years in other countries. The questionnaire is only sent to at least third year students, but we have kept the options for the first two years just in case the word of mouth reaches them. The 4+ students are those who still study after four years of standard curriculum because they still have to complete the courses or internship.

The replies were divided evenly among the students and the graduates. Having the same number of two types helped us to see if there are any differences between recent graduates and students without being biased against one type.

The second question is **"How competent do you feel in programming?"** and wants to find out how they evaluate themselves about the most important and needed technical skill [6]. The answers are a list of multiple choices: ''Not competent at all,'' ''a little competent,'' ''average,'' ''good,'' and ''very good''. The most chosen answers are ''average'' and ''good'' with 31 and 27 replies, respectively. Only 7 replies say that they are ''very good,'' and 10 of them find themselves ''a little competent''. The final option was chosen by 3 students, who find themselves ''not competent at all.'' This answer shows that most students believe that they have acquired this skill, either during education, or with hands-on experience.

The next question is **"How many software projects have you developed?"** which is expected to correlate with the previous question. The answers are multiple choice, with answers 0, ''1 to 3,'' ''4 to 6,'' ''7 or more''. The most common answers are ''1 to 3'' and ''4 to 6'' with 30 answers each. There are 15 replies for ''7 or more,'' and 3 replies for zero projects. The replies to ''1 to 3'' and ''4 to 6'' are in good correlation with the number of replies to ''average'' and ''good'' of the previous question. The answers indicate that the more projects they complete, the more they feel competent at programming.

The fourth question wants to find out the hardest challenges they faced in a list of technical tasks and soft skills when the students have moved on from small projects to relatively larger ones. We have provided the following set of skills with the challenge level changing from 0: no experience, 1: easiest, up to 5: hardest. The list of tasks and skills, and their responses are

- Requirements analysis; most found it challenging at level 2 with 21 replies,
- Software Design; most found it challenging at level 4 with 21 replies,
- Software Development; most did not find it very challenging at level 2 with 26 replies,
- Software Tests; the replies are almost evenly distributed, however the most found it a challenge at level 2 with 18 replies,
- Project Management; most found it challenging at level 3 with 18 replies, with considerable number of replies to ''No experience'' with 16 replies,

- Personal Time Management; most found it challenging at level 2 with 21 replies,
- Communication with team members; most members did not find it challenging at all, with 30 replies at challenge level 1,
- Use of Project Tools (such as version control systems); most found it challenging at level 1 with 20 replies,

From another point of view, the hardest challenge was software tests, with 12 replies. The easiest challenge is found to be the communication with team members, with 30 replies.

The results are interesting; communication is surprisingly at a low challenge level, while software design is as challenging as it is expected to be. The results are further analyzed by grouping the student answers and graduate answers, and conducting a Mann-Whitney U test to determine if any of the skills are statistically significant between these groups. The tests are run by filtering out the replies for the ''No Experience'' choice, since its numeric value can alter the results, and without experience their contribution should be removed. As the results in Table 2 show, the replies to the ''Communication'' by the students and the graduates are significantly different. When the replies are analyzed group-wise, 19 graduates found communication with team members easy, while only 11 students had the same evaluation. Students have found it more challenging on levels 3, 4 and 5, with 8, 6, and 4 number of replies, respectively, while all these levels have 3 replies for the graduates. This also shows that, as the experience increases, communication soft skill also feels much easier.

TABLE 2. Results of the Mann-Whitney *U* test of the difference of replies to the fourth question between students and graduates. The results are acquired after removing the "No Experience" choice which is set to 0.

Tasks	U	p
Requirements Analysis	659.5	0.420
Software Design	765.0	0.259
Software Development	647.0	1.0
Software Tests	682.0	0.185
Project Management	476.0	0.965
Personal Time Management	558.0	0.652
Communication	475.0	**0.043**
Use of Project Tools	541.0	0.968

These results indicate that the new software engineers are still on their path to be different from students. From the students point of view, it is possible that the projects they consider as larger may still be relatively non-challenging compared to the industry standards, and they feel that they can handle these tasks.

The next questions asks **"In which of the following did you acquire more of your project development skills other than programming?"** with replies being ''with engineering education,'' and ''without engineering education''. The replies are 66.7% for learning them outside engineering education. The follow up multiple answer question asks how they acquired these skills if they were acquired during their engineering education. We have provided two answers along with an optional input for customized answers. The answers provided are ''in theoretical courses'' and ''in practical courses, or in courses with projects.'' The replies are overwhelmingly with the second option with 93.3%, as expected. The other follow up question is the same question for the answers who picked the ''without engineering education.'' We have provided multiple answers as ''by working with experienced engineers,'' ''by using resources (such as books, videos) on project management and software development,'' and ''by my personal effort.'' Most selected answer was the ''personal effort'' with 48 replies. The second one was ''using resources'' with 35 replies, and the ''experienced engineers'' is the third with only 19 replies. Another follow up question for those who have acquired their skills outside education was the kind of projects they have developed. The multiple answers and their replies are ''personal projects'' with 44, ''part-time or full-time job'' with 31, ''FOSS'' with 23, and ''freelance projects'' with 20 replies.

This part analyzes how the students and graduates acquire their skills and how much of it comes from education. It is not surprising to find out that personal efforts and projects have come on top. Alongside personal projects, the next most common way to acquire these skills are is to be involved in a part-time or full-time job.

The next question directly relates to the second question: **"How competent do you feel as a software engineer?"** with the same answers. More than half of the replies, 40 to be exact, have replied with the ''average'' option. Almost a quarter of the replies have chosen the ''good'' option. A relatively smaller percentage of 19.2% have chosen ''a little competent''. Two replies found themselves to be ''very good'' while three replies felt ''not competent at all''. When compared with the second question, which enquires about the self-evaluation of their programming skills, they feel more ''average'' as engineers, while they feel ''good'' for being competent in programming.

Another Mann-Whitney U test has been conducted to determine if there were differences in the graduates and students about their self-evaluation of their competence in programming and software engineering. The test show that, there is no significant difference between these groups for the programming competence ($U = 915.5, p = 0.102$), however, there is a statistically significant difference for their self-evaluation of competence in software engineering with a score of $U = 946, p = 0.044$.

The final question required an open ended answer for any remarks they wanted to make and only eight of them did so. The general highlights, as translated to English, are as follows;

- The projects at the university are way different from the projects in real life. Also, the projects are not complete because everyone in the team are novice and they try to manage the project and develop it at the same time.
- The project teams are not evenly balanced and an experienced student might take away the experience that could be gained by others by trying to do everything on their own.

TABLE 3. The questions and the number of replies to multiple choices and answers, except the third question.

Question	Type	Answers	Replies	Percentage
What is your year of study in the software engineering education program?	Multiple Choice	2	7	9
		3	12	15.4
		4	15	19.2
		4+	5	6.4
		Graduated	39	50
How competent do you feel in programming?	Multiple Choice	No competent	3	3.8
		A little	10	12.8
		Average	31	39.7
		Good	27	34.6
		Very good	7	9
How many software projects have you developed?	Multiple Choice	None	3	3.8
		1 to 3	30	38.5
		4 to 6	30	38.5
		7+	15	19.2
In which of the following did you acquire more of your project development skills other than programming?	Multiple Choice	With Engineering Education	26	66.7
		Without Engineering Education	52	33.3
How were these skills acquired during within engineering education?	Multiple Answers	In theoretical courses	4	8.9
		In practical courses, or in courses with projects	42	93.3
		Custom answer: Thesis	1	2.2
		Custom answer: On my own	2	4.4
		Custom answer: FOSS	1	2.2
How were these skills acquired outside engineering education?	Multiple Answers	By working with experienced engineers	19	29.7
		By using resources (such as books, videos) on project management and software development	35	54.7
		By personal effort	48	75
		Custom answer: GitHub	1	1.6
		Custom answer: Taking courses	1	1.6
What kind of projects have you developed outside engineering education?	Multiple Answers	Personal projects	44	69.8
		FOSS	23	36.5
		Freelance jobs	20	31.7
		Part time or full time job	31	49.2
		Custom answer: Internship	1	1.6
How competent do you feel as a software engineer?	Multiple Choice	No competent	3	3.8
		A little	15	19.2
		Average	40	51.3
		Good	18	23.1
		Very good	2	2.6

TABLE 4. The number of replies given for every choice in the question "How challenging were the following when you have moved from simple projects to relatively larger ones?" The answer represent challenge levels, 1 being the easiest and 5 the most challenging. Level 0 represents no experience.

Tasks	0	1	2	3	4	5
Requirements	9	15	21	19	12	2
Software Design	5	13	16	17	21	6
Software Development	6	15	26	17	12	2
Software Tests	10	9	18	14	15	12
Project Management	16	15	13	18	6	10
Personal Time Management	9	18	21	14	10	6
Communication	6	30	15	11	9	7
Project tools	12	20	19	16	5	6

The second item brings a lot to the table; team projects have their pros and cons. The comment refers to the cons; how can we provide an environment where the students can improve their soft skills such as team work and communication? There are a lot of factors, such as balancing the team members abilities and grading individual effort. There are studies on increasing team work efficiency, but it requires a longitudinal study to extract their effect on the gap.

The results of the questionnaire are in alignment with existing literature. The study by Falkner *et al.* reports that self-regulated learning skill help students to achieve their goals [29], just as the students have reported to have picked up their skills by personal effort. For the students and recent graduates who have completed the questionnaire, the communication skill has been found to be significantly different, which is an evidence that recent graduates have found themselves lacking in soft skills, similar to existing studies [48], [51]. Several solutions are suggested in Table 1.

B. INTERVIEWS WITH STUDENTS AND RECENT GRADUATES

We have interviewed thirteen students and recent graduates. The interviews are mostly conducted via e-mail with the following questions in the first part, and a set of follow-up

questions in the second part. Face-to-face interviews took around 30 minutes to 90 minutes, depending on the answers of the respondent.

The first set of questions warm up the respondent by letting them consider their own personal process first and track it to team projects and soft skills.

1) What is your personal software development process?
2) If the problem at hand is a large software project, how does it change your personal process?
3) Even though the project is developed with a team and a software development process, you will be doing the development on your own; how does your process change then?
4) If you are given the chance to pick your software development team for a software project in a course, what would be your selection criteria?
5) Software development requires skills alongside programming, such as teamwork and communication. How do you think you acquired these skills?

The personal process is mostly not necessary for our purposes, however, contrasting it with the second question helps us identify how their process changes when they move on to a relatively larger project. The answers point out that they mostly spend more time on design, requirements, or both to handle the project. In one interview, the respondent remarked that he wasn't always involved with individual projects before he gained experience in large projects, and his large-scale development experience was reflected on his personal software development process, and he didn't feel any overhead.

The team selection question is asked to check if the students value soft skills. Not surprisingly, almost all respondents emphasized that the team members should have good communication skills, should be a team player and that they should be motivated and be willing to contribute. Some have asked for technical skills, reasoning that these skills matter for the success of the project.

Acquiring and improving skills is a skill within itself and the respondents have acquired or improved their skills in various ways. Some remarked that they have improved their soft skills after being employed professionally, some emphasized self learning, and some mentioned that the university has provided them with great opportunities to develop these skills.

After these sets of questions, we have directed these follow-up questions to the respondent who either have graduated or have experience in developing real life projects.

1) When you were a student, how hard was it for you to design and implement relatively larger programs? Do you remember the challenges you have faced, and how you overcome those challenges?
2) When you were a student, were the practical courses enough? How much time did you invest in your own self study?
3) Do you think that the projects you were assigned were similar to real life projects? In what aspects were they different?
4) Did your experiences in course team projects help you to develop your soft skills?
5) Did you have any problems when you got your first job? If yes, what were they? If no, why do you think you did not have any problems?

The first question directly asks what the second question in the first set of questions wants to ask. The most common answer is the difficulty of the design process of the software. One interviewee remembers that she didn't know where to start, and that designing and putting everything together was the most challenging of all. She overcame the problem by researching on her own, and remarked that practical courses helped in this regard. Another challenge emphasized by respondents is collaboration, since no one in the team had experience of running a project as a team.

The answers to the second question agree that while the course hours are adequate, they still had to invest a lot of time in their personal development because the projects are either not interesting or small in scale. All the answers to the third question stated that the course projects were not like real life projects, because they were small in scale and were very well defined in a controlled environment.

The respondents felt very differently for the course team projects. Some have remarked that they have not acquired any skill because the group was focused on the grades, rather than the project. Some have learned important life lessons, such as bad things could happen in a real life project, too. Some say that they have improved their soft skills. One student claimed that their project team had acquired the software development process so well that he was able to point out the mistakes in the process in his job. Consequently, the projects were full of hits and misses.

The most important question is the one that asks if they have faced any problems when they got their first job. More experienced graduates expressed that they did not have any problems. Others noted that they did not have any experience in working on large projects so they had problems with the configuration tools of the system, and figuring out the tasks the software should perform on different components of the system.

Several points made by the students are also reflected in existing literature. The students, at least for the course projects, are aware of the importance and value of the soft skills, as in studies [48], [51]. These projects are also evaluated on their realism, which is a common source of the gap, as discussed in [5], [37]. Also, self-learning has been mentioned once again, which provided the students a good opportunity to sharpen their skills [29]. Proposed solutions are discussed in Section III-B2, as well as listed in Table 1.

Overall, we have come to the conclusion that having experience during education clearly helps the student to narrow or close the gap. It is also beneficial to the students if they figure out how to learn on their own while they are studying to earn their degree.

C. INTERVIEWS WITH ACADEMICS

It is a regular task for the academics to review their courses at the end of each semester, and the curricula at the end of each academic year. Therefore, they are already aware of several problems that are associated with the gap. We have interviewed six academics from four different universities to find out their views on the following questions. The interviews were mostly done online. Face-to-face interviews took around half an hour to one hour. We have compiled their replies to the following questions to provide a list of their suggestions and observations. While it would be generous to generalize these replies, we believe that they have correlations with the limited existing literature on the perspective of the professors, one being the study by Pinto *et al.* [35].

1) What are the problems that you face in the projects and homework you assign to your students?
2) Can you think of anything that could be done to better prepare the student for the industry?
3) Do you think the students get their competency in the profession during the courses, or in the projects they involve in outside of courses? How competent do you think they are when they graduate?

One common theme the academics see in the projects and homework assignments in reply to question 1 is the inability of the students to articulate their work in their reports. Communication is an essential soft skill, and its two basic forms, written and spoken communication, is severely lacking. Another problem is that the students are not interested in programming; many times, a professor remarks, even though it will bring them extra points, they refrain from implementing the bonus section of an assignment that adds an improvement to the program. It should be noted that one student also complained about this, however, from their point of view, the student stated that most of the projects were not interesting at all, and like him, most of his friends did not feel like working on it anymore.

The academia has always been aware of the gap. They have put forward several methods, as mentioned previously, to incorporate industrial experience into their courses in reply to question 2. Our respondents have come up with the following suggestions to prepare the students for the industry.

The most obvious answer, as expected, is to have collaborations with the industry. A professor put forward the idea of extending the internship: starting in summer and continuing through the following semester as a part-time job. The job should be part of a course that provides credits towards the graduation of the student. One pointed out that the companies that offer internships should also provide financial compensation. Another proposition was that a guest from the industry should be invited to project presentations to get their comments on the code and the project. The academics did also mention that the students should have experience working on an existing code base.

On the final question, the academics see eye to eye that the students should improve their skills more by involvement in real life projects. Students are more motivated to complete these projects because they are either compensated financially, or because they volunteer for the task.

As a conclusion, the academics agree that industry must be represented more than just the internship, the students should be involved in real life projects, and the academics should choose the projects in their courses not for only the comprehension of the topic, but also as an interesting real life problem.

D. INDUSTRY REQUIREMENTS

We have interviewed the students, recent graduates, and the academics. In order to get an idea of what the industry wants, we considered the current job applications available online. A search in a popular employment website in Turkey for the position of ''software engineer'' returns 144 results. Out of the 144 results, we have randomly reviewed fifty up-to-date advertisements. Table 5 and Table 6 show the desired technical and soft skills, respectively.

TABLE 5. The number of occurrences of hard skills sought by the companies from the list of 50 randomly selected software engineering advertisements out of 144.

Skills	Occurrences
At least a B.S. degree	50
Programming	50
Software Engineering	32
Database	25
Experience (1 to 2 years)	15
Cloud Computing	14
Testing	10
Version Control System	7
Domain specific knowledge	6
Network	5
Operating Systems	4

TABLE 6. The number of occurrences of soft skills sought by companies from the list of 50 randomly selected software engineering advertisements out of 144.

Skills	Occurrences
English	49
Team work	29
Problem solving	28
Analytical thinking	27
Self learning	24
Communication	22
Writing	16
Responsibility	12
Good attitude	4

As the list for the number of occurrences of hard skills show, all of the positions require at least a B.S. degree in software engineering or related disciplines, such as computer engineering. All advertisements require proficiency in a programming language; the list aggregates them all under ''Programming.'' A large portion of the advertisements require comprehension of software engineering concepts, such as the software processes. Another important item in the list is that 15 of them look for at least 1 or 2 years of experience.

On the table for the number of occurrences of soft skills, proficiency in the English language is almost always required. The advertisements emphasize this requirement for the candidate to be able to stay up-to-date with the current technologies. The runner-up is Team Work, since more than half of the advertisements list this skill as required. The other skills that are listed alongside team work are problem solving and analytical thinking which are required to overcome several unforeseen problems. Self learning and communication are also required by the advertisements.

There are studies that analyze the needs of the industry either by reviewing advertisements [59] or by surveying the practitioners in the industry [60]. Lethbridge has surveyed 200 software developers and managers on about 75 educational topics and wanted to find out how important each topic is to their career. The results show that the participants believe the programming related topics are the most important. Lethbridge states that this might be because of the developers tendency to start implementation immediately without extensive requirements analysis or design, or because they spend most of their time in development and other tasks are distributed to various members of the team. Soft skills are also high on the list as the developers indicate the importance of technical writing and giving presentations. Radermacher *et al.* have interviewed 23 personnel from software companies to find out the areas in which the recent graduates struggle the most [61]. The participants have reported that the recent graduates lack project experience, which is also reflected in their usage of software tools. The new employees have reported that they have not either not used the software tool at all, or did not use it in a production environment. They also report that the graduates find it hard to communicate with their colleagues and customers.

While our own survey brings out the results in Turkey, it has apparent correlations with the findings reported in existing literature. These statistics and the related work in the literature show that the industry value the software engineering degree, because it gives the candidate a set of technical skills that make them desirable in the employment market. The industry clearly wants to stay up-to-date with the current technologies, thus makes English a top priority in the advertisements. The other soft skills are in alignment with the data we gathered from the existing studies; team work, communication and self learning make it to the list with large shares in the requirements.

The results also show that the academia and the students should focus on the soft skill set. While a degree in software engineering and comprehension of related concepts are achievable by graduating from the program, the soft skills might still require additional effort for acquisition and improvement.

V. MIND THE GAP

We have analyzed the gap from various perspectives along with the views from the industry, academia and the software engineers. When all related entities are studied together, it is possible to see that the causes of the gap are distributed among all of them. In the current state of the profession, the gap is inevitable and it depends on all of the entities involved.

By considering all entities, our study provides a unique opportunity to access all so that each can find out what they can do to acknowledge and narrow the gap.

A. THE SOFTWARE INDUSTRY

From the perspective of the industry, the recent graduates are not competent enough to do well in the real life projects. They lack in the usage of current tools, they have poor soft skills, and they struggle at the beginning of their career. Therefore, the industry favors the students with high GPA, so that they can quickly train them, or the students who have experience because they have been involved in several projects, commercial or open source.

While they are right to look for competent engineers, they should also be aware that the software industry, unlike any other industry, is relatively young and agile. As new technologies arrive, new platforms are created that require new methodologies, programming languages, software distribution channels, and so on. The industry, therefore, has the following options.

- Favor recent graduates with high GPA so that they can be trained before or after they are employed, or graduates with experience in commercial or FOSS projects, and graduates that have the motivation and the self-learning skill. This will require a thoroughly scrutinized interview process. More details are provided in Section III-A1 and in Section III-B3.
- Get involved in the education of professionals via organizations such as IEEE, or via the advisory boards of universities, to set the software engineering program outcomes. This was proposed by Beckman *et al.* in [10] and is detailed in Section I.
- Invest or get involved in course projects so that they can be developed by students who need real life experience. Similar propositions were made by the professors we have interviewed in Section IV-C, as well as in studies such as [37].
- Maintain ties with the academia by accepting students via internship, and help train students with current tools.
- Be aware that with the current speed of advancements, some methodologies and techniques require time to reach the academic curriculum. Publish and share the experiences gained, so that it would be easier to gain the attention of the academia.

We believe that the industry is the main driving force in creating the gap, and the academia and the students are trying to catch up with them in the practicalities of the profession. They should also be aware that while the academia aims to deliver students that have strong understanding of the underlying and enduring principles of software engineering, they rely on the continued learning skills of students for new technologies that come up.

B. THE ACADEMIA

The academic perspective has a lot of challenges; the most apparent one is to balance the curricula with courses that can improve both the hard and soft skills of students. The academia has recognized the gap early on and tried to adapt to close or narrow it. While it has been successful in some aspects, the constant advances in the profession has kept the gap open for three decades. The academia may consider the following options to keep an eye on the gap and do its own share to narrow and close it.

- The curricula of software engineering programs should consider different approaches in teaching and learning, such as problem-based learning. Several cases are reported in [21]–[23], [46] as discussed in Section III-A2.
- The courses in the curriculum can have strong links between them, such as backward and forward references, so that the students can see the larger picture, and recognize their connection. Such holistic approaches are discussed in Section III-A2, and in studies such as [19], [20], [47].
- The academics should be involved in industrial projects to keep up with the new methodologies in the profession.
- Industrial contacts can be invited to project exhibitions and presentations. Similar suggestions were made by professors in our interviews, as discussed in Section IV-C.
- The realism in course projects can be provided by inviting real customers, collaborating with the industry, or by getting involved in FOSS projects. Involving FOSS is a common approach and is applied in several cases [33]–[36] as discussed in Section III-B2 along with other approaches.
- The academics should come up with interesting projects to attract students so that they would be motivated to work on them. This option has two sides to it; the student's point of view is discussed in Section IV-B, and the academics point of view is discussed in Section IV-C.
- The academia should promote the program as technical and sociotechnical practice. This is discussed in detail in Section III-B1.
- The academia should accept a diverse range of prospective students to remedy the technical male stereotype of software engineers, as discussed in Section III-B1.

We agree with Craige *et al.* in their view that the aim of academia should not be vocational training, but a middle ground can be found to balance the fundamental principles with the practicalities of the profession [5]. As will be discussed, the middle ground can only be found with the help of all entities.

C. THE STUDENTS AND RECENT GRADUATES

The perspective of the students and recent graduates is the most important, since they are the entities most affected by the gap. The case for students is more challenging because it is possible that they will not be aware of the gap until they start their internship at a company. Therefore, we believe that our study would benefit the students the most. It provides an overview of the profession and a detailed analysis of the gap which would give them a heads up before their graduation.

Recent graduates will be well aware of the gap once they are employed. Depending on their career, they will need to keep their skills sharp by continuously learning new technologies.

Both the students and recent graduates have the following options to keep the gap at a minimum.

- Be aware of the gap and invest in developing their skills by getting involved in commercial or FOSS projects to improve hard and soft skills. This option has been analyzed in Section III-A3.
- Acquire the self-learning skill and appreciate the importance of continued learning, while having a strong knowledge in the basic principles of software engineering. Self learning has been emphasized in SEEK [4], as discussed in Section III-A3. The interviews with students also bring out its importance in Section IV-B.
- Be motivated to contribute to projects and realize that students and graduates will benefit in the long run.

Although this list looks brief compared to other entities, their work requires the most effort. The students are not as experienced as the entities in the industry or the academia, and they need to invest a lot of time and effort to become proficient in software engineering. To their advantage, it is relatively easier than other engineering disciplines to gain experience during their education. We believe that gaining the self-learning skills early on would be greatly advantageous in the long run, since the profession constantly faces new advances.

D. MINDING THE GAP

It can be noticed that it is a challenge to talk about the gap without considering other entities involved. Therefore, we would like to express how these entities relate and can work together to handle the gap.

The industry, being aware of the gap, should improve its involvement with the academia, passing as much information as possible about the latest methodologies. The market inclinations are vital in the determination of technologies that gain popularity and the industry can help itself and the academia to be proactive in this regard. On the other hand, the academia can strengthen its ties to the industry by academic collaboration of projects and therefore creating an environment for real life experience for its students. This will balance the fine line between fundamental principles and the practicalities of the profession.

The academia has immediate access to students and graduates. The academics can come up with interesting projects that make use of current platforms and methodologies that would attract the attention of students, while the students can motive themselves to complete the projects even though they know that they are toy projects, investing for a time in the near future to reap the benefits. The academia can be the binding

agent between the students and the industry if and when they have stronger connections to it.

Engineering is a social activity and software engineering is no exception. The academia should also promote the software engineering program as a sociotechnical profession and try to appeal to a wider range of prospective students to change the introvert male stereotype of engineers. This promotion can also help existing students to be more social so that they can compete with their rivals for job positions.

The students can turn their disadvantage of lack of experience into an advantage if they are aware of the current status of the profession and the gap. Acknowledging the gap, their first priority should be gaining experience to improve their hard and soft skills, so that they can apply for an industrial software engineering position upon their graduation. During their education, they have several options to be involved in real life projects, such as FOSS, and learn the latest tools and platforms while improving their knowledge in basic software engineering principles. The students will be observing the requirements of the industry, the sociotechnical promotion of the profession, how the hard and soft skills are used in a team project, and the constant need of self-learning as new technologies emerge. We believe that these will change their perception of the profession and will help the most essential entity of the gap, students and recent graduates, to take action to close it as much as possible until their graduation. Nevertheless, this requires the efforts of other entities as well, and we believe that it will not be possible for all students to be able to achieve it.

VI. CONCLUSION

In this study, we have analyzed the gap from the point of view of three major groups, the industry, the academia, and SSE. We have considered the gap from their own perspectives, and also listed the causes that we have observed and that were also reported in the literature.

A questionnaire is conducted with the students and the recent graduates to find out the magnitude of challenges they have faced when they have moved to relatively larger projects. We have reported and analyzed its results and found out that although there is no difference between the two groups statistically, there is a difference that is statistically significant to their replies of the challenge of communication, and of how competent they feel as software engineers.

We have interviewed students and recent graduates either online or face-to-face to extract more information about these challenges. This resulted in the conclusion that experience is the best cure to narrow the gap. They have reported that even though they had some challenges when they moved to relatively larger projects, they have overcome them as they had more experience. The interview with the academics was also very fruitful because they have provided novel ideas for narrowing the gap, such as improving industrial collaboration and providing more engaging projects for the students to develop.

The job advertisement survey shows us that the companies look for candidates who can keep up with the current state of the profession, and therefore require proficiency in the English language and the ability to learn on their own. They are also looking for engineers who can work as a member of the team and solve the problems they face on their own. The technical skills are not much different; a degree in software engineering and programming is always required. Proficiency in software engineering concepts are also a sought skill.

Finally we have considered all entities on their own, as well as together, as to how they can acknowledge and close the gap, and propose a list of suggestions for all.

We conclude that the gap is not going to close any time soon, at least for some of the recent graduates. The industry will innovate new practicalities and the academia needs to balance the hard and soft skills with realistic projects. It requires constant attention and hard work for all of the entities involved. Therefore, we extend the suggestions for the students to all of the entities in the field: we all should be aware of the gap, and to keep it at a minimum we should be on alert for new technologies, learn to embrace that change, adapt to it, and be successful as the technological advances take us to new platforms.

ACKNOWLEDGMENT

We would like to thank the professors, graduates, and students who have shared their sincere opinions and views on the matter. Our wholehearted thanks go to Lecturer Ilker Korkmaz for pointing out structural improvements, to Prof. Dr. A. Sermet Anagun for his guidance on the statistical tests, and to Lecturer Ralph John Berney for his extensive proofreading and language editing. We are also greatly indebted to the anonymous reviewers for their valuable comments and feedback which have improved the manuscript significantly.

REFERENCES

[1] P. Kruchten, "Putting the 'engineering' into 'software engineering,'" in *Proc. Austral. Softw. Eng. Conf.*, Apr. 2004, pp. 2–8.

[2] S. Yu and S. Zhou, "A survey on metric of software complexity," in *Proc. 2nd IEEE Int. Conf. Inf. Manage. Eng.*, Apr. 2010, pp. 352–356.

[3] G. Matturro, F. Raschetti, and C. Fontán, "Soft skills in software development teams: A survey of the points of view of team leaders and team members," in *Proc. IEEE/ACM 8th Int. Workshop Cooperat. Hum. Aspects Softw. Eng.*, May 2015, pp. 101–104.

[4] *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, Joint Task Force Comput. Curricula, New York, NY, USA, 2015.

[5] M. Craig, P. Conrad, D. Lynch, N. Lee, and L. Anthony, "Listening to early career software developers," *J. Comput. Sci. Colleges*, vol. 33, pp. 138–149, Apr. 2018.

[6] C. S. Miller and L. Dettori, "Employers' perspectives on it learning outcomes," in *Proc. 9th ACM SIGITE Conf. Inf. Technol. Educ. (SIGITE)*, Cincinnati, OH, USA, 2008, pp. 213–218.

[7] K. Alboaouh, "The gap between engineering schools and industry: A strategic initiative," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2018, pp. 1–6.

[8] E. Goold, "Engineering students' perceptions of their preparation for engineering practice," in *Proc. 6th Res. Eng. Educ. Symp.*, Dublin, Ireland, 2015, pp. 1–9.

[9] G. A. Ford and N. E. Gibbs, "A master of software engineering curriculum: Recommendations from the software engineering Institute," *Computer*, vol. 22, no. 9, pp. 59–71, Sep. 1989.
[10] H. Beckman, N. Coulter, S. Khajenoori, and N. R. Mead, "Collaborations: Closing the industry-academia gap," *IEEE Softw.*, vol. 14, no. 6, pp. 49–57, Nov. 1997.
[11] M. Shaw, "Software engineering education: A roadmap," in *Proc. Conf. Future Softw. Eng. (ICSE)*, 2000, pp. 371–380.
[12] D. L. Parnas, "Software engineering programs are not computer science programs," *IEEE Softw.*, vol. 16, no. 6, pp. 19–30, Nov. 1999.
[13] A. Begel and B. Simon, "Struggles of new college graduates in their first software development job," in *Proc. 39th SIGCSE Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2008, pp. 226–230.
[14] P. Bourque and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK (R)): Version 3.0*, 3rd ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 2014.
[15] E. Brechner, "Things they would not teach me of in college: What microsoft developers learn later," in *Proc. Companion 18th Annu. ACM SIGPLAN Conf. Object-Oriented Program., Syst., Lang., Appl. (OOPSLA)*, 2003, pp. 134–136.
[16] E. Tuzun, H. Erdogmus, and I. G. Ozbilgin, "Are computer science and engineering graduates ready for the software industry?: Experiences from an industrial student training program," in *Proc. 40th Int. Conf. Softw. Eng., Softw. Eng. Educ. Training (ICSE-SEET)*, 2018, pp. 68–77.
[17] G. Samarthyam, G. Suryanarayana, A. K. Gupta, and R. Nambiar, "Focus: An adaptation of a swebok-based curriculum for industry requirements," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 1215–1224.
[18] J. Yang and J. Wang, "Review on free and open source software," in *Proc. IEEE Int. Conf. Service Oper. Logistics, Inform.*, vol. 1, Oct. 2008, pp. 1044–1049.
[19] A. A. Maciejewski, T. W. Chen, Z. S. Byrne, M. A. De Miranda, L. B. S. Mcmeeking, B. M. Notaros, A. Pezeshki, S. Roy, A. M. Leland, M. D. Reese, A. H. Rosales, T. J. Siller, R. F. Toftness, and O. Notaros, "A holistic approach to transforming undergraduate electrical engineering education," *IEEE Access*, vol. 5, pp. 8148–8161, Mar. 2017.
[20] M. Rashid and I. A. Tasadduq, "Holistic development of computer engineering curricula using y-chart methodology," *IEEE Trans. Educ.*, vol. 57, no. 3, pp. 193–200, Aug. 2014.
[21] M. M. Silva, S. R. G. dos Santos, J. C. R. da Silva, L. A. V. Dias, and A. M. da Cunha, "Problem based learning a practical approach for software engineering interdisciplinary teaching," in *Proc. 8th Int. Conf. Inf. Technol., New Gener.*, Apr. 2011, pp. 1046–1047.
[22] Y. R. Pena, J. R. H. Hernández, and M. L. Vazquez, "Problem-based learning. An experience on the inclusion of quality problems in educational software engineering," in *Proc. World Eng. Educ. Forum-Global Eng. Deans Council (WEEF-GEDC)*, Nov. 2018, pp. 1–6.
[23] I. Richardson and Y. Delaney, "Problem based learning in the software engineering classroom," in *Proc. 22nd Conf. Softw. Eng. Educ. Training*, Feb. 2009, pp. 174–181.
[24] T. Nurkkala and S. Brandle, "Software studio: Teaching professional software engineering," in *Proc. 42nd ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2011, pp. 153–158.
[25] C. N. Bull, J. Whittle, and L. Cruickshank, "Studios in software engineering education: Towards an evaluable model," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 1063–1072.
[26] D. Rosca, "Acquiring professional software engineering skills through studio-based learning," in *Proc. 17th Int. Conf. Inf. Technol. Based Higher Edu. Training (ITHET)*, Apr. 2018, pp. 1–6.
[27] C. N. Bull and J. Whittle, "Supporting reflective practice in software engineering education through a studio-based approach," *IEEE Softw.*, vol. 31, no. 4, pp. 44–50, Jul. 2014.
[28] M. Luukkainen, A. Vihavainen, and T. Vikberg, "Three years of design-based research to reform a software engineering curriculum," in *Proc. 13th Annu. Conf. Inf. Technol. Educ. (SIGITE)*, 2012, pp. 209–214.
[29] K. Falkner, R. Vivian, and N. J. Falkner, "Identifying computer science self-regulated learning strategies," in *Proc. Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, 2014, pp. 291–296.
[30] M. Marques and S. F. Ochoa, "Improving teamwork in students software projects," in *Proc. IEEE 27th Conf. Softw. Eng. Educ. Training (CSEE T)*, Apr. 2014, pp. 99–108.
[31] L. F. Capretz and F. Ahmed, "A call to promote soft skills in software engineering," *Psychol. Cogn. Sci. J.*, vol. 4, pp. e1–e3, Aug. 2018.
[32] Y. Sedelmaier and D. Landes, "SWEBOS—The software engineering body of skills," *Int. J. Eng. Pedagogy*, vol. 5, pp. 20–26, Feb. 2015.
[33] D. M. C. Nascimento, C. F. G. Chavez, and R. A. Bittencourt, "The adoption of open source projects in engineering education: A real software development experience," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2018, pp. 1–9.
[34] Z. Hu, Y. Song, and E. F. Gehringer, "Open-source software in class: Students' common mistakes," in *Proc. 40th Int. Conf. Softw. Eng., Softw. Eng. Educ. Training (ICSE-SEET)*, 2018, pp. 40–48.
[35] G. H. L. Pinto, F. F. Filho, I. Steinmacher, and M. A. Gerosa, "Training software engineers using open-source software: The professors' perspective," in *Proc. IEEE 30th Conf. Softw. Eng. Educ. Training (CSEE T)*, Nov. 2017, pp. 117–121.
[36] T. M. Smith, R. McCartney, S. S. Gokhale, and L. C. Kaczmarczyk, "Selecting open source software projects to teach software engineering," in *Proc. 45th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2014, pp. 397–402.
[37] R. Simpson and T. Storer, "Experimenting with realism in software engineering team projects: An experience report," in *Proc. IEEE 30th Conf. Softw. Eng. Educ. Training (CSEE T)*, Nov. 2017, pp. 87–96.
[38] C. Szabo, "Student projects are not throwaways: Teaching practical software maintenance in a software engineering course," in *Proc. 45th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2014, pp. 55–60.
[39] I. Richardson, L. Reid, S. B. Seidman, B. Pattinson, and Y. Delaney, "Educating software engineers of the future: Software quality research through problem-based learning," in *Proc. 24th IEEE-CS Conf. Softw. Eng. Educ. Training (CSEE T)*, May 2011, pp. 91–100.
[40] D. Graziotin and P. Abrahamsson, "Making sense out of a jungle of JavaScript frameworks," in *Product-Focused Software Process Improvement* (Lecture Notes in Computer Science), J. Heidrich, M. Oivo, A. Jedlitschka, and M. T. Baldassarre, Eds. Berlin, Germany: Springer, 2013, pp. 334–337.
[41] F. S. Ocariza, K. Pattabiraman, and A. Mesbah, "Detecting Inconsistencies in JavaScript MVC applications," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, vol. 1, May 2015, pp. 325–335.
[42] K. Peguero, N. Zhang, and X. Cheng, "An empirical study of the framework impact on the security of JavaScript Web applications," in *Proc. Int. World Wide Web Conf. Steering Committee, Companion Web Conf.*, Lyon, France, 2018, pp. 753–758.
[43] A. Pano, D. Graziotin, and P. Abrahamsson, "Factors and actors leading to the adoption of a JavaScript framework," *Empirical Softw. Eng.*, vol. 23, pp. 3503–3534, Dec. 2018.
[44] M. Rashid, "System level approach for computer engineering education," *Int. J. Eng. Educ.*, vol. 31, no. 1, pp. 141–153, 2015.
[45] M. Rashid, "A methodology for the development of competencies required by industry," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 3, pp. 7851–7856, Sep. 2014.
[46] V. Ribaud and P. Saliou, "The cost of problem-based learning: An example in information systems engineering," in *Proc. 26th Int. Conf. Softw. Eng. Educ. Training (CSEE T)*, May 2013, pp. 259–263.
[47] H. J. C. Ellis, R. A. Morelli, T. R. D. Lanerolle, and G. W. Hislop, "Holistic software engineering education based on a humanitarian open source project," in *Proc. 20th Conf. Softw. Eng. Educ. Training (CSEE T)*, Jul. 2007, pp. 327–335.
[48] F. Ahmed, L. F. Capretz, S. Bouktif, and P. Campbell, "Soft skills and software development: A reflection from the software industry," Jul. 2015, *arXiv:1507.06873*. [Online]. Available: https://arxiv.org/abs/1507.06873
[49] C. Turhan and I. Akman, "Employability of IT graduates from the industry's perspective: A case study in Turkey," *Asia Pacific Educ. Rev.*, vol. 14, pp. 523–536, Dec. 2013.
[50] R. Baldwin, D. J. Finch, M. Zehner, and L. K. Hamilton, "An exploratory study of factors affecting undergraduate employability," *Educ. Training*, vol. 55, pp. 681–704, Sep. 2013.
[51] A. Radermacher and G. Walia, "Gaps between industry expectations and the abilities of graduates," in *Proc. 44th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2013, pp. 525–530.
[52] A. M. Moreno, M. I. Sanchez-Segura, F. Medina-Dominguez, and L. Carvajal, "Balancing software engineering education and industrial needs," *J. Syst. Softw.*, vol. 85, no. 7, pp. 1607–1620, Jul. 2012.
[53] D. Joseph, S. Ang, R. H. L. Chang, and S. A. Slaughter, "Practical intelligence in IT: Assessing soft skills of IT professionals," *Commun. ACM*, vol. 53, pp. 149–154, Feb. 2010.
[54] L. F. Capretz, "Bringing the human factor to software engineering," *IEEE Softw.*, vol. 31, no. 2, p. 104, Mar. 2014.

[55] M. Bailey, C. Baillie, J. Impagliazzo, D. M. Riley, and G. D. Catalano, "Special session-not many women in engineering—So why should i care? Bridging gender gaps and stereotypes," in *Proc. 36th Annu. Conf. Frontiers Educ.*, Oct. 2006, p. 1.
[56] B. Benedict, D. Verdin, R. Baker, A. Godwin, and A. Thielmeyer, "I don't FIT the stereotype, but i see myself as an engineer: First-year engineering students' attitudes and beliefs about their engineering identities," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2018, pp. 1–7.
[57] R. McCord, "Bazinga! You're an engineer. . . you're_! A qualitative study on the media and perceptions of engineers," in *Proc. ASEE Annu. Conf. Expo.*, 2013, pp. 1–20.
[58] E. A. Eschenbach, M. Virnoche, E. M. Cashman, S. M. Lord, and M. M. Camacho, "Proven practices that can reduce stereotype threat in engineering education: A literature review," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2014, pp. 1–9.
[59] F. Gurcan and C. Kose, "Analysis of software engineering industry needs and trends: Implications for education," *Int. J. Eng. Educ.*, vol. 33, no. 4, pp. 1361–1368, Feb. 2017.
[60] T. C. Lethbridge, "Priorities for the education and training of software engineers," *J. Syst. Softw.*, vol. 53, no. 1, pp. 53–71, 2000.
[61] A. Radermacher, G. Walia, and D. Knudson, "Investigating the skill gap between graduating students and industry expectations," in *Proc. Companion 36th Int. Conf. Softw. Eng. (ICSE)*, 2014, pp. 291–300.

DAMLA OGUZ received the B.S. degrees in software engineering and industrial systems engineering and the M.S. degree in computer engineering from the Izmir University of Economics, in 2008, 2009, and 2012, respectively, and the Ph.D. degree from joint Paul Sabatier University, France, and Ege University, Turkey, in 2017.

She was a Research Assistant with the Department of Computer Engineering, Izmir Institute of Technology, from 2009 to 2017. Since 2017, she has been an Assistant Professor with the Department of Software Engineering, Yasar University, Turkey. Her research interests include query optimization in large-scale distributed environments, software engineering education, and data mining.

KAYA OGUZ (M'10) received the B.S. degree in software engineering from the Izmir University of Economics, Izmir, Turkey, in 2007, the M.S. degree in computer games technology from the University of Abertay Dundee, Scotland, U.K., in 2009 (for which he received the scholarship from the Izmir University of Economics), and the Ph.D. degree in information technologies from Ege University, Izmir, Turkey, in 2016.

He was a Research Assistant with the Izmir University of Economics, from 2009 to 2011. He was a Lecturer with Ege University. He has taught several courses with the Software and Computer Engineering Department as a Guest Lecturer, Izmir University of Economics. In 2017, he has joined the Department of Computer Engineering, Izmir University of Economics, as an Assistant Professor. His research interests include graph theory, procedural content generation for computer games, software engineering education, and medical imaging analysis.

Dr. Oguz has been a member of ACM, since 2010.