Competencies for Code Review

PAVLÍNA WURZEL GONÇALVES, University of Zurich, Switzerland
GÜL ÇALIKLI, University of Glasgow, United Kingdom
ALEXANDER SEREBRENIK, Eindhoven University of Technology, The Netherlands
ALBERTO BACCHELLI, University of Zurich, Switzerland

Peer code review is a widely practiced software engineering process in which software developers collaboratively evaluate and improve source code quality. Whether developers can perform good reviews depends on whether they have sufficient competence and experience. However, the knowledge of what competencies developers need to execute code review is currently limited, thus hindering, for example, the creation of effective support tools and training strategies. To address this gap, we firstly identified 27 competencies relevant to performing code review through expert validation. Later, we conducted an online survey with 105 reviewers to rank these competencies along four dimensions: frequency of usage, importance, proficiency, and desire of reviewers to improve in that competency. The survey shows that *technical competencies* are considered essential to performing reviews and that respondents feel generally confident in their technical proficiency. Moreover, reviewers feel less confident in how to communicate clearly and give constructive feedback - competencies they consider like-wise an essential part of reviewing. Therefore, research and education should focus in more detail on how to support and develop reviewers' potential to communicate effectively during reviews. In the paper, we also discuss further implications for training, code review performance assessment, and reviewers of different experience level.
Data and materials: https://doi.org/10.5281/zenodo.7401313

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

Additional Key Words and Phrases: Code Review, Competency, Skills, Training, Human Factors

ACM Reference Format:
Pavlína Wurzel Gonçalves, Gül Çalıklı, Alexander Serebrenik, and Alberto Bacchelli. 2023. Competencies for Code Review. *Proc. ACM Hum.-Comput. Interact.* 7, CSCW1, Article 38 (April 2023), 33 pages. https://doi.org/10.1145/3579471

1 INTRODUCTION

Code review is a collaborative activity in which software developers evaluate source code. In its most widespread form [56], code review is used to evaluate and discuss newly submitted code changes to improve their quality, catch potential functional or maintainability issues, and decide on whether to integrate them into the production code [2, 5, 19, 44].

Code reviews can act as a sort of training process where knowledge transfer takes place [5, 66]: Educators use code reviews to improve students' coding skills, program comprehension abilities, knowledge of coding standards, and peer communication [66]. In addition, a similar process of

Authors' addresses: Pavlína Wurzel Gonçalves, p.goncalves@ifi.uzh.ch, University of Zurich, Binzmühlestrasse 14, Zurich, ZH, Switzerland, 8050; Gül Çalıklı, HandanGul.Calikli@glasgow.ac.uk, University of Glasgow, Glasgow, Scotland, United Kingdom; Alexander Serebrenik, a.serebrenik@tue.nl, Eindhoven University of Technology, Eindhoven, North Brabant, The Netherlands; Alberto Bacchelli, bacchelli@ifi.uzh.ch, University of Zurich, Binzmühlestrasse 14, Zurich, ZH, Switzerland, 8050.

2573-0142/2023/4-ART38
https://doi.org/10.1145/3579471

improving skills through code reviews takes place during professional life [5] where companies and Open Source projects use code reviews as a means to integrate and mentor new team members [59].

Reviewing experience is essential for an effective code review [35]. For instance, between the 20th and 50th month of developers' experience on a project, acceptance of their code changes becomes more likely [53]; whereas developers' first contributions are more likely to be rejected [64]. Furthermore, the usefulness of comments in code review rapidly increases in the first year of tenure [11], suggesting that developers gaining code review experience also improve their competence.

It is crucial to identify competencies needed for code review to design code review training [9] and support and assess code review performance. Yet, various competencies needed to conduct code review are not available in the literature [4]. Several frameworks in the literature describe the competencies needed for software development [42, 57, 71]. However, such generic competency models are often insufficient for specific software engineering roles or tasks [15, 60, 60, 72] such as code reviews.

Developing competency models is costly for research resources [72]. Therefore, in this study we use a systematic mapping study by Assyne et al. [4] to identify existing frameworks for software engineering competencies [42, 57, 71] that we adjust for the code review context using the method of expert validation [30]. We further prioritize the identified competencies with developers' self-reported perspectives and experience to understand which competencies developers consider more essential and which they need to improve in to become better reviewers.

Through interviewing experts, we have defined a set of 27 competencies that are relevant to performing code reviews well. The final list consisted of 12 technical, 11 social, and four personal competencies in six clusters: (1) understanding the change, (2) systematically navigating the code base, (3) assessing the software quality, (4) discussing informatics topics, (5) relating to colleagues, and (6) demonstrating personal competencies such as review time-management and managing priorities.

Subsequently, we conducted an online survey with 105 reviewers. Their answers allow us to rank the final list of competencies based on how frequently they need a competency, how important they consider the competency to perform code reviews well, how proficient they are in that competency, and which competencies they would like to improve. We also analyzed how these rankings differ between novice and expert code reviewers.

Through our analysis, we identified that technical competencies are the most used and important. Reviewers also feel the most confident in their technical competence. Communicating in an understandable way and giving constructive feedback are on top of the list of frequently used and important competencies as well. Yet, reviewers feel less confident in these social competencies and they would like to improve accordingly. Interestingly, current research and tooling focus mainly on supporting developers' technical competence for code reviews, even though developers have more issues with social competencies.

Furthermore, we have identified that novice and expert reviewers would like to improve on different competencies. On the one hand, novice reviewers prefer to improve competencies useful to understand and better contextualize the code change; on the other hand, expert reviewers are more interested in improving competencies for communication, collaboration, and mentoring.

While these results can help prioritize which competencies can be used to design focused training activities for novice or expert reviewers, the list of competencies itself can be used to facilitate discussion in teams about the code review process and their specific training needs, or to hint in which areas developers need tool support. Furthermore, competencies can be used as an alternative way to assess code review performance.

2 RELATED WORK

In this section, we describe the current code review practice and present the theoretical concepts related to competencies, the context in which competencies are going to be used throughout the paper, how competencies can be used in practice, and how skills and competencies are represented in the Software Engineering literature.

2.1 Modern Code Review

Code review is a widely used software engineering practice [5, 29, 56, 59], which is recognized as a valuable tool for reducing software defects and improving software projects' quality [2, 5]. In a typical code review, software developers collaborate to evaluate changes before they are integrated into a project's code base. Examples of such code changes can be (1) including new features, (2) making behavioral changes, improvements, and optimizations to an existing code, (3) bug fixes and rollbacks, or (4) refactoring and large-scale changes[31]. In practice, a developer (author) submits a code change to be inspected by one or more developers (reviewers), who can decide to include the change in production or not. Before including the change, reviewers can also ask the author for clarifications and code improvements. As a response to reviewers' comments, the author can upload new versions of the code change, thus creating an iterative process that finishes when the reviewers are either satisfied with the change (and include it in production) or reject it [19, 49]. Throughout this paper, we use 'author' for the developer who submitted the change, 'reviewer' for the developer who is reviewing changes, and 'developer' for any developer that could at some point participate in reviews.

Code review can become expensive and time-consuming [16]. Therefore, it is useful to identify the challenges developers face in this task. Identifying defects and understanding the submitted change is mentally challenging for developers [5, 8, 49]. To understand the change well, developers need to process a vast amount of information and evaluate the change and its rationale within the context of an entire software system [49]. Not only do developers need to understand the change itself, but they also need to understand each other [5]. Code reviews are commonplace for confusion [22] or interpersonal conflicts [24, 74] that can stand in the way of an efficient review.

Lack of skills is another challenge that can hinder code review performance [3, 41]. Competency-based Education and Training (CBET) can provide a basis to aid developers in improving their competencies and, consequentially, their code review performance and practice [9]. Designing CBET tailored for code reviews can only be done once the competencies that developers specifically need to conduct code reviews are known.

2.2 Competencies

Understanding what attributes people need to perform well at work has been of great interest in work psychology. Commonly these attributes were understood as Knowledge, Skills, and Abilities (KSAs) [46, 61]. Using KSAs to describe the worker requirements enables a more fine-grained depiction of what is required and expected from a worker to perform well in a specific position. However, successful performance at work is related to many more concepts that determine workers' behavior, such as personality traits, attitudes, and values. Therefore, a more applied concept of *competency* has emerged to overcome these shortcomings [46]. Competencies—as described by Bartram [7]—can be understood as behavioral repertoires that workers can apply to achieve the desired work goal. With this definition, the focus shifts from workers' characteristics to workers' desired behavior. The concept of 'behavior' does not carry assumptions about workers as people or personalities. Furthermore, behavior can be more easily described, observed, and trained.

The use of competencies is done in many fields and contexts. Competencies can be position-specific or general – applying to a broad range of jobs and positions. General competencies such as *cooperating* [7] might be an essential aspect of code reviews. However, providing a useful depiction of competencies needed for code review might need a more specialized or fine-grained scope, for example, to describe specific social interactions like *mentoring* new team members or technical skills like *code comprehension*.

2.3 Competency Modelling

Defining the skills and competencies of knowledge workers is especially difficult because their behavior and process cannot be directly observed and, instead, can be described as a set of cognitive processes. There are specific methods such as the Cognitive Task Analysis [72] that derive competencies through various methods – observation, interviews, or questionnaires – that enable to understand the individual cognitive process when performing the task.

In the last years, recruitment and performance assessment in organizations shifted to using *competency modeling* to describe behaviors essential to performing a job [38]. The main advantage is the success of competency modeling in creating organization-wide or even market-wide integration of strategies for recruitment, training, and assessing workers' potential and performance. Once the competencies that a developer needs are defined, they can be used to define job advertising, recruit developers with the needed competencies, define software developers' improvement needs, design training focused on such needs, and assess code review performance. If code review requires code comprehension, this competency can be tested when recruiting, measured when assessing code review performance, and re-assessed before and after training focused on improving code comprehension.

An example of a training approach that utilizes competency models is Competence-based Education and Training (CBET) [9]. It requires integrating a specified set of competencies into authentic learning activities that address core problems and dilemmas of the professional practice. Designing appropriate training activities requires identifying these core problems, for instance, by interviewing experts and educators. Therefore, describing the competencies needed for code reviews is the first step and a prerequisite to designing focused training programs. Our study aims to identify such a comprehensive set of competencies – behaviors and cognitive processes – to distinguish better and worse code review performance that can have wide use for developers and organizations.

2.4 Skills and Competencies for Software Development

In Software Engineering, developers' competence is approached from various angles – as knowledge, expertise, or skill. The knowledge and technical expertise required for software engineers are collected in the Software Engineering Body of Knowledge (SWEBOK) [1]. SWEBOK describes ten knowledge areas, which also serve as a basis to investigate the competencies that software engineers need to perform well at work [57]. However, SWEBOK does not cover soft skills in depth. The EVELIN (Experimental improVEment of Learning software engINeering) project [63] aims to analyze the needed skills and propose a Software Engineering Body of Skills (SWEBOS) [63]. SWEBOK and SWEBOS serve as a basis to better design and inform higher education for software engineers. Skills in Open Source Software (OSS) development have also been viewed through the lens of expertise and experience with specific projects, programming languages, or APIs [20].

Apart from these essential resources in Software Engineering, other studies have investigated more specific sets of skills and competencies that can be used to understand the needed competencies not only for Software Engineers in general but also for more specific software development roles [4]. However, none of these works can be directly applied to the context of code review.

Describing skills and competence does not serve only to demonstrate what knowledge and expertise are useful to perform tasks and roles but also to distinguish different levels of competence or better and worse performance. The SECAT (Software Engineering Competency Assessment Tool) proposes three such levels [62]. At the first level, software engineering students are expected to achieve *functional competence* - to know that specific goals, functionalities, and operability need to be achieved and to be able to process the technical information. In the second level, students are expected to achieve *process competence* - to understand and evaluate the appropriateness of solutions, cost-benefit ratios, the usability of solutions, or to demonstrate problem awareness and the ability to abstract. In the last level, students should achieve the *holistic shaping competence* and demonstrate sensitivity to context, find creative solutions and develop personal competencies for communicating with others and organizing their work efficiently.

Indeed, IT professionals improve their skill set throughout their careers, with the technical competencies being highest in the Senior Software Engineer roles and the non-technical competencies being the most developed on the management level [17]. Furthermore, the non-technical competencies gain their importance throughout IT professionals' careers over the technical ones [39]. Therefore, understanding which competencies require more development for students and young professionals and which are needed more by senior roles can give valuable insights into what training each group requires to improve their work, performance and productivity. This study also aims at understanding the different competencies between more novice and expert reviewers.

3 METHODOLOGY

This section presents our research questions and describes the employed study design, data analysis, and sample of respondents. This study has been approved by the ethical committees of the authors' institutions. Additional files are available in the supplementary material at https://doi.org/10.5281/zenodo.7401313.

3.1 Research Questions

We aim to identify what competencies are necessary for code review. Knowing these competencies is a fundamental step to inform tool design, education, training, and performance assessment to support developers' potential during code reviews. Furthermore, we aim to investigate which competencies play a more critical role, as a way to focus on future research and education efforts.

When describing competencies, we are aligned to the following dimensions commonly used for competency assessment: [27, 43]: (1) *usage* frequency (*i.e.*, how often competencies are used in a specific role or activity), (2) *importance* (*i.e.*, how important the competencies are for good performance), and (3) *proficiency level* (*i.e.*, what level of proficiency in these competencies is needed to perform the task(s)). We also aim to assess which competencies need further improvement (*i.e.*, *preference to improve* competencies). In this study, we achieve this by gathering developers' experiences, perceptions, and opinions.

Competencies for Code Review. To explore and prioritize the competencies that are essential to perform reviews well, we consider two perspectives from the literature, competencies' usage frequency and importance for good performance [27, 43].

- $RQ_{1.1}$: *What competencies do reviewers report using more frequently?*
- $RQ_{1.2}$: *What competencies do reviewers consider more important?*

Proficiency Level and Improvement of Competencies. We aim to identify which competencies to focus on while training and supporting reviewers by identifying which competencies reviewers lack ($RQ_{2.1}$) and which competencies they would like to improve ($RQ_{2.1}$).

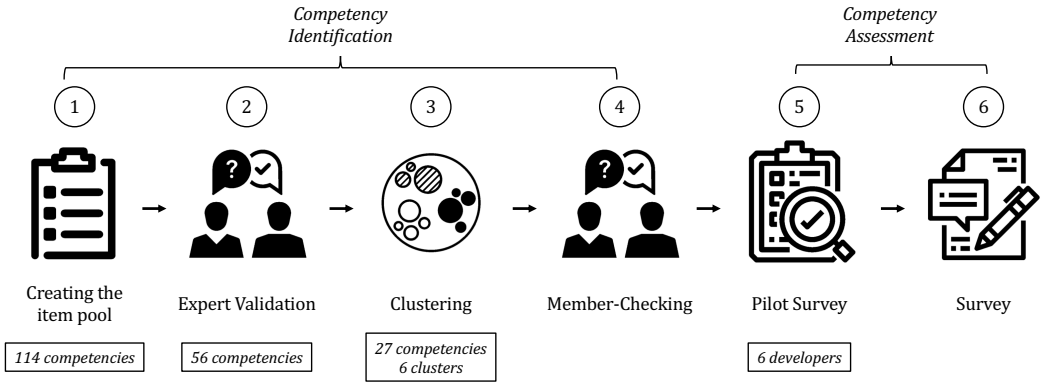


Fig. 1. The Study Design Steps.

- $RQ_{2.1}$: *What level of competencies for code review do developers report to have?*
- $RQ_{2.2}$: *Which competencies would developers like to improve?*

Competencies and Experience. Distinguishing which competencies are the core ones throughout a person's career, and which ones become more salient or fade out can give important insights into the different development needs of novice vs. experienced reviewers [73]. Experience with code reviews is important for performing effective reviews [35]. Therefore, we expect that novice and experienced reviewers might consider different competencies as essential and want to improve different ones. We collect evidence on these differences in the aforementioned dimensions (*i.e.*, *usage* frequency, *importance*, *proficiency level*, and *preference to improve*).

- $RQ_{3.1}$: *Do novice and experienced reviewers use different competencies during code review?*
- $RQ_{3.2}$: *Are the competencies that novice and experienced developers consider important different?*
- $RQ_{3.3}$: *Do novice and experienced reviewers report a different proficiency level of the competencies needed for code review?*
- $RQ_{3.4}$: *Would novice and experienced reviewers like to develop a different set of competencies?*

To address these RQs, we designed and carried out a two-phase study (as depicted in Figure 1): (Phase 1) - Competency identification and (Phase 2) - Competency assessment.

3.2 Competency Identification

To identify the competencies needed for code reviews, we derived an initial item pool from competency models and frameworks relevant to software development and engineering roles (see Step 1 in Figure 1). The initial item pool was validated by experts in a round of interviews (Step 2 in Figure 1). The expert validation led to further refinement of the proposed competencies. These competencies were rephrased and adjusted according to the feedback of experts (Step 3). Furthermore, we clustered the resulting competencies by their meaning to create a survey-length list of competencies as described later in this section. After the adjustments, we presented the final, refined set of competencies for code review to the experts for validation (*e.g.*, in terms of relevance and adjustments made to the initial item pool) (Step 4). The changes in the number of items/competencies in each Step of the study is described in Table 1. Examples of adjustments to the items throughout the refinement of the list of competencies are presented in Table 2.

3.2.1 Creating the initial item pool. Competency frameworks or competencies needed for code reviews are not defined in the literature [4]. However, competency frameworks for software

Table 1. Number of Competencies in the Item Pool After the End of Each Study Step

Study Step	Number of Competencies	Technical	Social	Personal
Creating the Item Pool	114	58	25	31
Content Expert Validation	58	31	16	9
Refinement and Clustering	27	12	11	4
Member-Checking	27	12	11	4

Table 2. Examples of Adjustments to the Items

Study Step	Original item	Modification	Reason	Adjusted item
Content Expert Validation	Know and use tools (IDEs)	Rephrasing	Each team uses different tools for viewing code reviews, editing the code, etc.	Proficient use of the tools used in the team
	Develop mental models	Rephrasing	The mental model relevant for code review is specifically the mental model of the given code change	Develop mental model of the change
Clustering	Able to evaluate and ensure consistency throughout code base and requirements	Splitting	Consistency of the code base is one of the important aspects of evolvability of the system	Evaluate the code change quality with respect to its effects on the evolvability of the software system
			Another item already covered evaluating requirements with regards to the code change	Evaluate if implementation fits requirements
Member-Checking	Resolve conflicts	Adding definition and rephrasing	Competency unclear to the expert. Definition supported by literature on conflict resolution competence.	To handle conflicts - Enhancing productive outcomes of a conflict while minimizing its escalation or harm done.

development in general and for specific roles (*e.g.*, requirement analysts and tester [34, 60]) are available. We cannot directly apply general software development skill sets such as the one by Moreno et al. [45] to code review because they cover a broader range of activities and provide a level of abstraction that is too high to create focused training or discuss detailed development needs for the context of code review. Similarly, the competency frameworks for specific roles formulate finer skills, but they are tuned to a different set of activities than code reviews.

Hence, to develop a set of competencies for code reviews that can be presented in a survey, we selected several competency frameworks for general software development to refine them into a more specific competency list. We have used the systematic mapping study by Assyne et al. [4] to identify appropriate models based on the following selection criteria: (1) the study or framework defines a specific list of competencies or skills for software development, (2) these competencies create a comprehensive set needed for a task or role (including those related to the technical competence but also the social or personal one) on a matching level of abstraction, (3) these competencies were derived from a thorough analysis of interviews with practitioners in software development, as would be done by the traditional methods of job analysis [72]. Furthermore, competencies should describe behavior patterns that can be trained or developed to perform better [73]. Therefore, studies describing competencies and skills by knowledge or tools (*e.g.*, technology stacks) were excluded from the study. Through this filtering, we selected the following frameworks:

- *Essential competencies of software engineers* by Turley and Bieman [71]. This study identified four groups of competencies essential for software engineers: (1) *task accomplishment competencies*, such as methodical problem solving, good scheduling and estimation, or the active seeking of necessary training and knowledge; (2) *personal attribute competencies* like pride in quality and productivity, perseverance or desire to improve things; (3) *situational skills* such as emphasis on elegant and simple solutions and focus on user/customer needs; (4) *interpersonal skills* like seeking help or willingness to confront others.
- *Competency Framework for Software Engineers* by Rivera-Ibarra et al. [57]. This framework represents a set of technical, social, and personal competencies for software engineers: (1) the *technical competencies* cover the analytical and learning ability in knowledge areas of software engineering inspired by SWEBOK [1] as well as the need to evaluate, select, adapt and use appropriate technological tools; (2) *social skills* refer to the aptitude to connect with others, cooperate in teams, and handle and resolve conflicts. (3) *personal skills* include skills to act, work, and develop as a professional as well as to keep and protect one's limits.
- *Competence model for informatics modeling and system comprehension* by Linck et al. [42]. It is a model for computer science education, specifically informatics modeling and system comprehension. Unlike the previous models, this model details primarily technical and cognitive competencies needed for software development. There are five main dimensions of competencies: (1) *selection and application* of appropriate informatics systems in diverse contexts, including abilities like structuring problems or transferring application fields. (2) *system comprehension* through analyzing and evaluating system requirements, systematically exploring and testing the systems, or knowing and evaluating algorithms and data structures. (3) *system development*, which is a dimension detailing knowledge and skills required for developing and re-engineering informatics systems such as knowing and applying design patterns, software development process models, or documenting source code. (4) *dealing with system complexity*, which is a competence dimension referring to measuring and evaluating system complexity and dealing with processing and working with complex information. (5) *non-cognitive skills* that address the competencies related to cooperation, communication, and motivational skills, but also the ability to perceive and anticipate the effects of systems.

All the competencies from the selected models have been split into technical, social, and personal competencies. The competencies duplicated among the models were included only once in the initial item pool. The final amount of items included at this step was 114 (see Table 1).

3.2.2 Expert Content Validation. The initial item pool was discussed with four experts in an interview for content validation. Using experts for validating item pools is commonly done with

Table 3. Code Review Related Experience of Experts Included in Expert Validation

Expert	Field of Experience	Years of Experience
E1	Consulting	9
	Research	9
E2	Research	8
	Closed-source software development	13
E3	Closed-source software development	8
	Open-source sofware development	13
E4	Closed-source software development	7
	Open-source sofware development	1

two to ten experts [30]. In software engineering, experts should be selected based on qualitative criteria [40]. Therefore, for the expert content validation step, we selected experts from a varied range of code review-related expertise through personal contacts, i.e., we performed convenience sampling. The description of the experts interviewed in this study can be found in Table 3. Expert 1 has nearly a decade long experience in research and consulting. They conducted and published several large-scale studies at a major software company to understand and improve code reviews for various software teams. They worked for several companies to advance their code review products and also conducted workshops, coaching, and consulting for companies and teams performing code reviews. Expert 2 runs a software development company that performs code reviews. They also published research focused on improving code review tools to support how developers perform code reviews and on the adoption of code reviews in companies. Expert 3 has several years long experience with code reviews in software development in major software companies around the globe and from well-known open-source projects. Expert 4 has performed code reviews as a software developer in several mid-sized companies across Europe, mostly in the closed-source environment. Furthermore, we used quantitative criteria that could be used in the survey as well. While the mere length of experience is not very strongly related to performance, exceptional experts in various fields need typically around ten years of preparation to reach their peak performance [25]. Given that software development and code review commonly require a degree from higher education or further qualifications and training before performing code reviews, we consider a software developer with more than five years of experience performing code reviews (or five years in other relevant areas connected to code review for the content validation) to be an expert.

When involving experts for content validation, the items are usually rated on their relevance, style (simplicity, clarity, or ambiguity), and comprehensiveness in covering the domain of interest [52, 75]. In our interviews, the experts were first asked to list competencies and skills that they find important for code review. Secondly, the experts rated each item in the item pool as (not) relevant for code review, suggested revision of the proposed items and their definitions, and pointed out the lack of clarity. Thirdly, the experts were asked about the comprehensiveness of the entire set of competencies [75].

At the end of this step, we shortened the initial item pool to a list consisting of competencies marked as relevant by at least three out of the four experts. In this way, the list was shortened to 58 competencies (see Table 1). Furthermore, the experts provided feedback for the items that were used in the next Study Step - Refinement and Clustering (Section 3.2.3). While the list was covering all important areas according to the experts, the items needed several higher-level adjustments. In particular, the experts proposed to (1) formulate the competencies more in relation to the goals of

the code review, (2) formulate the skills and competencies as applied to *the software system* and *the change* that is being implemented, (3) merge and cluster some of the competencies, (4) avoid competencies related to activities that should be done before the review, such as architectural and design decisions, and (5) avoid competencies that are highly team-specific, such as the tools used for code review or software development.

3.2.3 Refinement and Clustering. With the feedback received from the experts, we have first rephrased the individual items according to the experts' general and item-specific feedback. Then, we merged items that experts deemed to be too similar or referring to the same area of competence. Furthermore, we excluded some items due to their poor fit to the definition of competence, such as cognitive resources or personality traits that are sometimes included in the literature due to a broader definition of 'skill' or 'competency'. Next, one of the authors clustered the competencies according to their similarity and relations as mentioned in the interviews. Another author reviewed the clustering and discussed and proposed changes to the phrasing of the competencies.

After this process, the rest of the authors reviewed the list of clustered competencies and proposed improvements to the clustering and formulation of individual competencies. At the end of this Study Step, the list of competencies has been finalized on 27 competencies, see Table 1.

3.2.4 Member Checking. The last step of the Competency Identification consisted of *member checking* [10]. We performed member-checking with the same experts who took part in the Expert Content Validation (Step 2 in Figure 1) to verify that we handled their feedback correctly and to mitigate researcher bias [10]. We later asked the experts to review the resulting list of competencies. The finalized list of competencies and the questionnaire sent to the experts is provided in the supplementary material. The experts rated the set of competencies as relevant, comprehensive, and understandable on a 5-point Likert scale ('Strongly Agree' to 'Strongly Disagree'). All answers were rated as 'Strongly Agree'. To obtain further feedback besides a quantitative one, we also asked the experts to provide detailed comments about the finalized list. Each expert provided detailed comments that led to rephrasing and refining competencies' definitions. In this step, we did not add/delete any competencies (see Table 2).

3.3 Using Self-report Methods to Investigate Competencies

Reviewing code is a cognitive activity taking place in the minds of reviewers who have to understand and evaluate the code. Even though this process leads to the creation of some observable artifacts (*e.g.*, code review comments and emails, which are used by developers to engage in a discussion), the cognitive process of reviewers is not observable directly. Currently, the cognitive processes taking place during code review are neither described nor understood in detail. Given these circumstances, self-report methods are the most valid ones to understand the competencies reviewers need to perform code review [72].

Nevertheless, we have used an existing dataset of review comments from three OSS projects [50] to investigate whether objective data could be used to corroborate and triangulate the results obtained by self-report methods. Code review comments are a data source frequently used by researchers [22, 50]. In our analysis, the first and second authors of this paper classified these comments independently with respect to the use of a certain competency. From each of the three projects, the raters (*i.e.*, the first and second authors) classified the first 50 comments excluding the comments by the code change's author. If the 50^{th} comment was part of a thread, the raters finished classifying the thread resulting in 102 classified reviewer comments. Each of the comments could be categorized as demonstrating multiple competencies. We used Krippendorff's alpha for multi-label annotation using MASI to calculate the inter-rater agreement resulting in $\alpha = 0.22$. This value means a very low agreement: the acceptable level is ≥ 0.8 [37].

During the classification of the code review comments, the following reasons led to the low inter-rater agreement: (1) the cognitive competencies cannot be observed directly, therefore, their classification is heavily dependent on the assumptions of the individual raters and such classification cannot be verified with the comments' authors, (2) cognitive processes that do not require interaction between the author and reviewer are unlikely to surface in the comments, (3) social competencies are often impossible to classify because the interpretation of these competencies depends on the motivation of the comment's author as well as on the interpretation of the receiver (*e.g.*, when a conflict is happening), (4) social competencies often need to be interpreted in the context of a whole thread rather than an individual comment, (5) some competencies are manifested by *not* writing comments (understanding the needed level of detail for the review might be a needed thought process which in the end leads to omitting to comment on certain issues and mentioning others), (6) the comments are descriptive (e.g., the reviewer commented on evolvability) while the competencies are prescriptive (*e.g.*, reviewers should know how to evaluate the evolvability issues).

These observations indicate that the classification of code review comments does not have sufficient concept validity to correctly depict the usage of competencies in code review. Therefore, this study further investigates competencies for code review solely through gathering reviewers' self-reported perspectives through a survey.

3.4 Survey

3.4.1 Structure and Measurement. To design the survey itself, we adhered to the best practices in software engineering survey research [32]. Figure 2 shows the survey structure (*i.e.*, order of the sections in the survey). At the beginning of the survey (*i.e.*, *Welcome* phase in Figure 2), we presented the developers with general information about the study and its purpose together with information on the survey's expected approximate completion time and data handling policy. The respondents needed to explicitly give consent so that their answers could be used in the study. After the *Welcome* phase, the developers were asked to respond to three of the questions proposed by Danilova et al. [18] to identify whether they are truly software developers (*Qualification questions* in Figure 2).

In the next sections of the survey, developers were presented with the full list of the competencies. They were asked to rate each competency in terms of usage frequency and proficiency level. Furthermore, we asked respondents to select (without ranking) up to five competencies they deemed to be most important and up to five competencies they would like to improve. Finally, we collected the participants' demographic data such as code review experience or highest achieved education level. We collected the demographic data at the end of the survey to reduce the stereotype threat [67].

In the survey itself, we used the term 'skill' instead of 'competency', as the former is a more common term used in the public space and potentially more understandable by practitioners, which is a good practice for performing research including a diverse audience [55]. On top of the aforementioned consent to process their data, at the end of the survey (*i.e.*, once they knew the survey's entire content and their answers) we asked participants whether we could include their responses in an anonymous public data set. We release this data set publicly together with the supplementary material.

To address the $RQ_{1.1}$ and to identify the essential competencies for code review, we measured a competency's usage frequency with a five-point Likert scale: 'Never', 'Less than Half of Reviews', 'About Half of Reviews', 'More than Half of Reviews' and 'Always'. Furthermore, the reviewers could select up to five most important competencies to answer $RQ_{1.2}$.

To address $RQ_{2.1}$, which asks what competencies reviewers need to develop, we measured the developers' proficiency level for each competency on a five-point Likert scale: 'Poor', 'Fair', 'Good',

Fig. 2. The Survey Structure.

'Very Good', and 'Excellent'. Reviewers also selected up to five skills they would personally like to improve providing the data to answer $RQ_{2.2}$.

3.4.2 Pilot. In Step 5 of the study design (see Figure 1), we conducted a pilot study with six developers experienced in performing code reviews. The pilot version of the survey had an additional section in which the pilot participants could rate the comprehensiveness, relevancy, and understandability of the set of competencies and survey questions. They could also indicate whether some survey questions were not clear and provide further comments on their experience. We made several changes after the pilot. For example, the option 'Not applicable' was added to the response anchors and one of the competencies was rephrased according to the participant's feedback. The pilot survey is available in the supplementary material.

3.5 Analysis

In $RQ_{1.1}$, we assess the usage frequency of a competency on a Likert-type scale from 'Never' to 'Always'. For the analysis, we employ the non-parametric variant of Scott-Knott's ESD test [69]. This variant of Scott-Knott's ESD test produced ranked groups of competencies in descending order with respect to competencies' usage frequencies. The Scott-Knott ESD has been successfully used in the past to analyze software engineering data [12, 13, 22, 68]. In the survey, developers were asked to choose up to five most important competencies. To answer $RQ_{1.2}$, we ranked each competency based on the total number of times the developers selected that competency to be in the five most important competencies. If multiple competencies received the same amount of responses, they were assigned the same rank and the following competencies were assigned a rank by skipping the rank that immediately follows this rank (Ranks 1, 2, 3, 3, 5, *etc.*).

We followed a similar procedure to answer $RQ_{2.1}$ and $RQ_{2.2}$ to indicate the developers' proficiency level of competencies for code review (measured on a 5-point Likert-type scale using anchors 'Poor' to 'Excellent'). We use the non-parametric version of the Scott-Knott's ESD algorithm [69] to provide ranked groups of competencies developers are more proficient with. Frequencies are used to report which competencies developers would like to improve the most/least.

$RQ_{3.1}$ to $RQ_{3.4}$ address the differences between novice and experienced programmers on the frequency, importance, proficiency, and desire for improvement concerning the investigated competencies. We assign the status of novice reviewer to a reviewer with less than five years of experience in performing code review and the status of expert reviewer to reviewers with more than five years of reviewing experience to be consistent with the definition of an expert from the step of expert validation. We produce rankings of competencies for novice and expert reviewers using the same methods as described in $RQ_{1.1}$, $RQ_{1.2}$, $RQ_{2.1}$ and $RQ_{2.2}$. We calculate Spearman's ρ to assess the relationship between the ranked groups of the competencies in the subset of novice and experienced reviewers. Furthermore, we perform the Mann-Whitney-U-test on the ordinal variables (frequency of usage and level of proficiency). The power analysis was performed to estimate the needed sample size for the Mann-Whitney-U-test, using an estimated effect size of 0.6 taken from a similar study on competencies [28]. The estimated sample size was calculated to be 98 participants. We decided against using p-value corrections, even though we perform multiple analyses of the same type. The

reason is that the analysis answers independent questions by using independent variables that do not contribute to an overarching hypothesis [33, 51].

3.5.1 Respondents. For the study, we have collected 105 complete and valid responses. The sample was recruited using several channels to ensure covering a diverse representation of performing code reviews in different contexts - by contacting companies, open source software projects, software development and code review related communities and platforms, and practitioners.

We contacted 50 companies from different countries and continents that currently advertised jobs requiring code reviews with the offer to create for them skills report of their employees using the anonymized data, two of which agreed to collaborate with us. Both were Europe-based. The first company employs 2,000 persons and the other one is a European branch of a 70,000-employee company. These companies have provided us with 15 respondents in total in exchange for a simplified report of the study results.

To reach open source developers, we asked core contributors and code owners of 20+ open source projects to share the study invitation within their project. Furthermore, we reached out to 20+ websites, Facebook pages, Twitter accounts, and Reddits related to software engineering, software development, and code reviews, some of which shared the invitation to the study. We also reached out to several practitioners developing code review tools or consulting in the code review area asking them to share our invitation within their network. Furthermore, we contacted individual software developers through advertising on the study authors' accounts on social networks: Twitter, Facebook, and LinkedIn.

The descriptive statistics of the sample are shown in Figure 3 and Figure 4. From the original number of 110 full responses, five were excluded due to the wrong answers to qualification questions (see Section 3.4.1). The final sample of 105 full responses consisted of reviewers performing reviews mostly at least once a week (N=86) and having more than 3 years of experience doing code reviews (N=89). Furthermore, most of the reviewers had experience performing reviews in the industry (N=92) but also in the open-source software development (N=57) and research (N=23). 44 reviewers were classified as novices (five years of code review experience or less) and 61 as expert reviewers (more than five years of code review experience). Therefore, our sample represents rather senior reviewers doing code reviews regularly and working in different settings of software development.

4 RESULTS

In the *Competencies Identification* phase of this study, we identified a list of competencies relevant to code review according to experts.

This list consists of 27 competencies in six clusters as presented in Table 4 and extended with the definitions we used in Appendix A. The technical competencies are organized in three clusters: *Change Understanding*, *System and Codebase Navigation*, and *Software Quality and Its Assessment*. The social competencies are represented by two clusters: *Discussing Informatics Topics* and *Relating to Colleagues*. The *personal* competencies form the last cluster.

4.1 Competencies for Code Review

In the $RQ_{1.1}$ and $RQ_{1.2}$, we investigated how often reviewers report using each of the identified competencies as well as which competencies they perceive as the most important. Using the survey answers, we ranked the competencies according to both dimensions. Table 4 summarizes the answers to $RQ_{1.1}$ and $RQ_{1.2}$. When discussing usage ($RQ_{1.1}$) for each competency, we report the most frequently indicated answer (mode), the number of respondents that have provided this answer, and the rank of the competency according to the Scott-Knott ESD algorithm. For importance ($RQ_{1.2}$), we report the percentage of respondents who have selected the competency as one of the five most

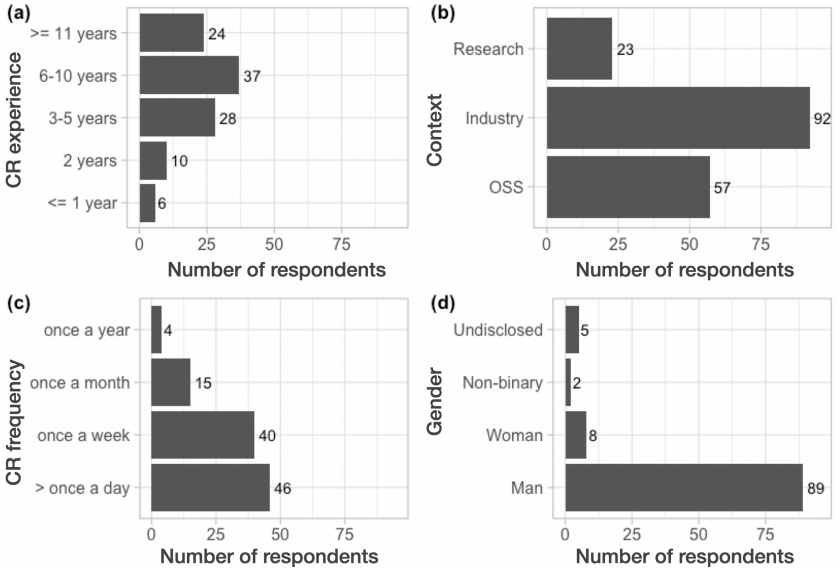


Fig. 3. Participants' demographics: **(a)** Years of Code Review (CR) experience ; **(b)** Context of performing CR ; **(c)** Frequency of performing CR ; **(d)** Gender.

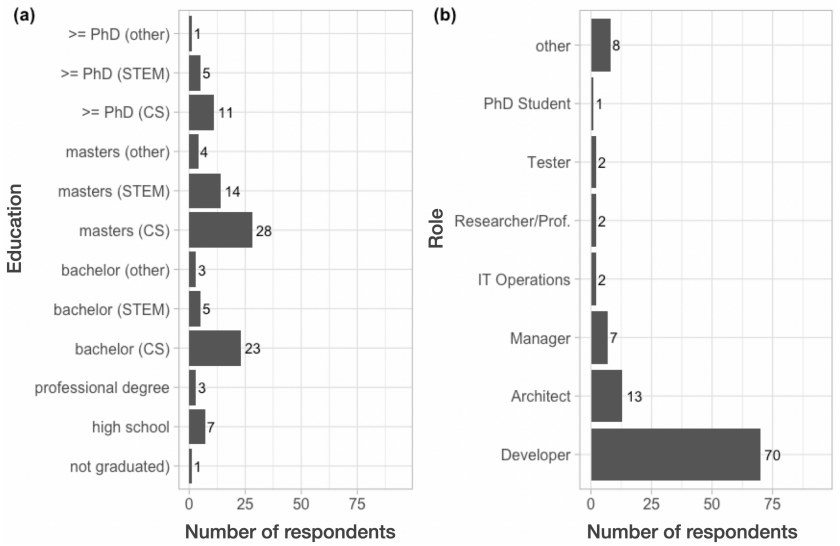


Fig. 4. Participants' demographics: **(a)** Highest achieved education; **(b)** Professional role.

important ones, as well as the rank determined by the percentage (the highest percentage ranked 1). Table 5 presents the summary of rankings of competencies in all the measured dimensions. For increased readability, Appendix B presents the lists of competencies sorted by their rankings.

The Scott-Knott ESD test identified ten ranked groups of competencies according to their usage frequency. In code review, reviewers report needing most frequently 'to communicate in an understandable way' (Group of Rank 1) followed by 'to give constructive feedback' (Group of Rank

2). These are both *social competencies*. Group 3 is formed by five, mostly technical, competencies. Developers report to often need the competency 'to develop a mental model of the change', 'to evaluate how the change fulfills requirements', 'to understand the purpose of the code review and the needed level of detail needed for it', 'to evaluate the effect of the change on the correctness of the software system', and 'to correctly use programming languages'. Even though the two most frequently used competencies are social ones, they are followed by the majority of the technical competencies, as depicted in Figure 5(a). The competencies that are reported as the least used are: 'to convince others about own ideas' and 'to effectively manage own emotions' (Group of Rank 8), 'to find relevant information beyond the code base' (Group of Rank 9), and 'to handle interpersonal conflicts' (Group of Rank 10).

Competencies that are more frequently used also tend to be considered more important by reviewers. There is a strong correlation between the two ($\rho(25) = 0.74$, $p = 0.02$). Similarly to the frequency of usage, the list is dominated by the technical competencies together with 'to communicate in an understandable way' and 'to give constructive feedback' (see Figure 5(b)). The five most important competencies for efficient code reviews are, according to reviewers, 'to develop a mental model of the code change' (selected among the most important by 59.0% of reviewers), 'to evaluate whether the implementation fulfills requirements' (53.33%) and 'to evaluate the code change quality with respect to its effects on the correctness of the software system' (45.71%). The last mentioned was evaluated by reviewers as the same level of importance as 'to give constructive feedback' (45.71%), followed by the need 'to communicate in an understandable way' (40.95%). Other competencies were considered among the most important by a lower portion of reviewers, as reported in Table 4. As the least important were considered 'to have an effective review time management' and 'to handle interpersonal conflicts' (both 1.90%).

4.2 Proficiency Level and Improvement of Competencies

$RQ_{2.1}$ and $RQ_{2.2}$ investigate how proficient reviewers rate themselves in the competencies and which competencies they would like to improve. Table 4 presents the mode of the individual competencies for the level of proficiency reported by respondents as well as the percentage of reviewers who would like to improve the competency. Table 5 summarizes the rankings of the competencies according to these dimensions.

Reviewers are quite confident about their competence. All modes for the level of proficiency are either 'Good' or 'Very Good'. Reviewers report the highest proficiency in being 'willing to learn and improve' and in their 'knowledge of programming languages'. Same as for frequency of usage and importance of competencies, there is a clear distinction between the technical and social competencies (Figure 5(c)). Reviewers report higher confidence in their technical competencies than in the social ones. Specifically, respondents report confidence in being able 'to systematically navigate codebase', 'to find relevant information beyond the codebase', 'to evaluate if implementation fulfills the requirements', and 'to understand the purpose of the code review and the level of detail needed for it'. Reviewers are the least confident 'to have effective review time-management', 'to handle interpersonal conflicts', and 'to adjust communication style to different individuals'. Reviewers are also not so confident in their ability 'to convince others about their ideas' and 'to effectively manage their own emotions'.

There is a moderate negative relationship between the perceived proficiency and what the respondents would like to improve to become more competent reviewers ($\rho(25) = -0.46$, $p = 0.01$). Unlike the previous dimensions, there is no clear pattern in which competencies developers would like to improve. They would like to work on all types of competencies (see Figure 5(d)). Respondents would like to mostly improve their ability 'to evaluate architectural implications of the change' (29.52%). Second, they would like 'to communicate in more understandable way' (27.62%).

Table 4. Comparison of Competencies on their Usage, Importance, Level of Proficiency, and Preference for Improvement

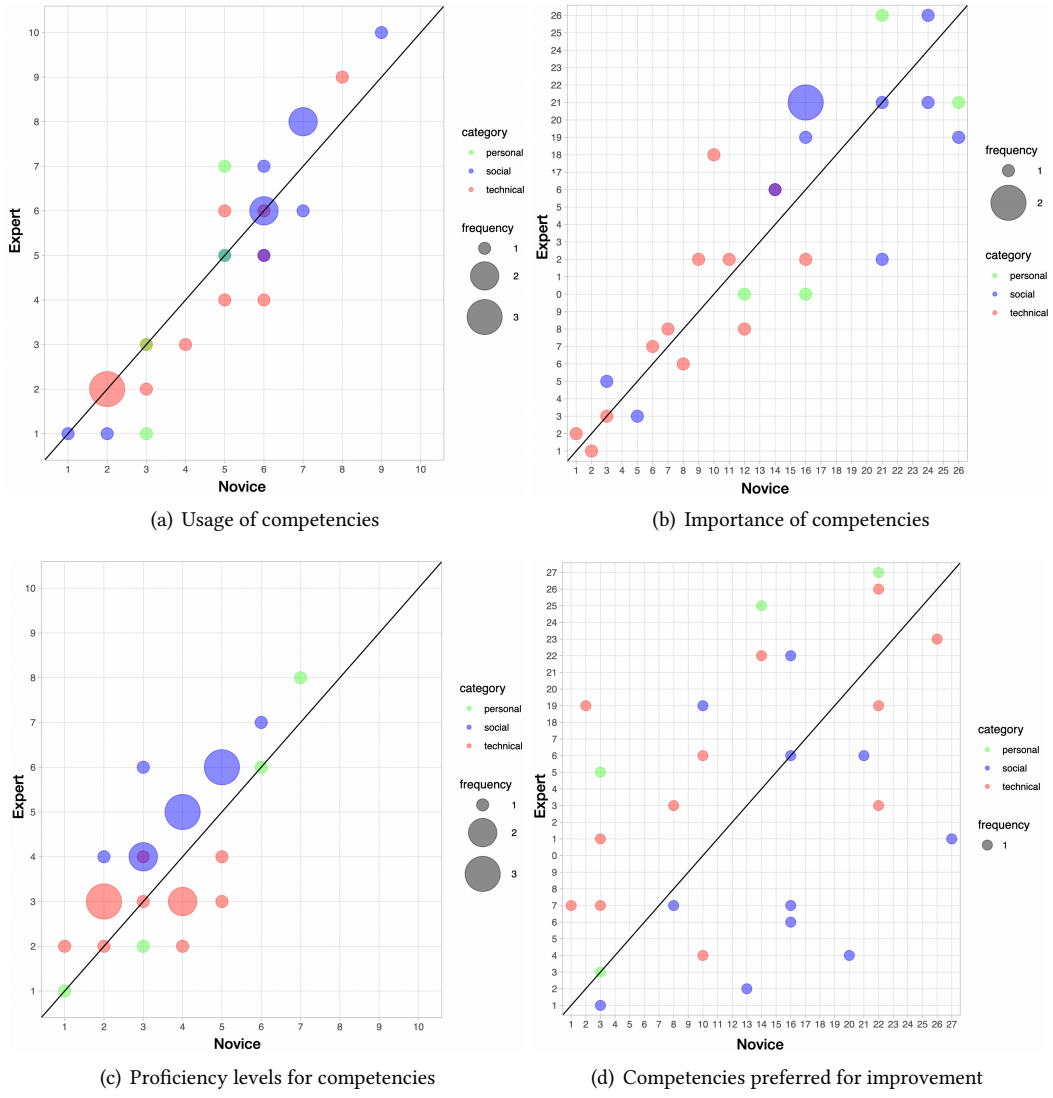
ID	Competency	RQ1					RQ2				
		Usage[1] $RQ_{1.1}$			Importance[2] $RQ_{1.2}$		Proficiency[3] $RQ_{2.1}$			Preference to improve[4] $RQ_{2.2}$	
	Technical	**Mode**	**N**	**Rank**	**%**	**Rank**	**Mode**	**N**	**Rank**	**%**	**Rank**
Cluster 1	*Change understanding*										
1	To develop a mental model of the change	Most	44	3	59.05	1	Very Good	42	3	18.10	10
Cluster 2	*System and codebase navigation*										
2	To systematically navigate codebase	Most	34	7	10.48	14	Very Good	44	2	9.52	22
3	To find relevant information beyond the codebase	Less than half	62	9	13.33	12	Very Good	45	2	12.38	17
4	To identify patterns in the code	Most	39	6	16.19	9	Very Good	46	3	15.24	15
5	To proficiently use the tools used in the team	Always	49	4	10.48	14	Very Good	44	2	6.67	25
Cluster 3	*Software quality and its assessment*										
6	To evaluate if the implementation fulfills the requirements	Always	59	3	53.33	2	Very Good	47	2	4.76	26
7	To evaluate architectural implications of the change	Most	39	6	31.43	6	Very Good	35	3	29.52	1
8	To evaluate the code change quality with respect to its effects on the correctness of the software system	Always	50	3	45.71	3	Very Good	51	3	20.95	6
9	To evaluate the code change quality with respect to its effects on the user-affecting properties of the software system	Most	39	5	16.19	9	Very Good	42	4	19.05	9
10	To evaluate the code change quality with respect to its effects on the evolvability of the software system	Most	45	4	30.48	7	Good	41	3	20.00	7
11	To estimate complexity of the problem and solution	Most	39	6	10.48	14	Very Good	45	3	21.90	5
12	To know and correctly use programming languages	Always	49	3	22.68	8	Very Good	42	1	10.48	21
	Social										
Cluster 4	*Discussing Informatics Topics*										
13	To communicate in an understandable way	Always	69	1	40.95	5	Very Good	38	3	27.62	2
14	To give constructive feedback	Always	52	2	45.71	3	Very Good	46	3	23.81	4
15	To understand the dynamics of the discussion and its conclusions	About half	31	6	2.86	23	Good	40	4	11.43	19
Cluster 5	*Relating to Colleagues*										
16	To adjust communication style to different individuals	Less than half	31	7	2.86	23	Good	31	6	16.19	14
17	To pick up and act on ideas of others	About half	32	7	3.81	22	Very Good	46	3	9.52	22
18	To be willing to compromise	About half	36	6	9.52	18	Very Good	33	4	12.38	17
19	To convince others about your ideas	About half/Less than half	37	8	4.76	20	Good	50	5	17.14	13
20	To mentor other developers	Most	31	7	10.48	14	Good	38	4	18.10	10
21	To effectively manage own emotions	Less than half	45	8	4.76	20	Good	34	5	14.29	16
22	To handle interpersonal conflicts	Less than half	54	10	1.90	26	Good	35	6	20.00	7
23	To be aware of the capacity of co-workers in terms of time, experience and knowledge	Most	30	6	5.71	19	Very Good	44	4	11.43	19
Cluster 6	**Personal**										
24	To have effective review time-management	Most	34	6	1.90	26	Good	36	7	25.71	3
25	To set, express, and manage priorities	Most	30	7	2.86	23	Good	49	6	18.10	10
26	To understand the purpose of the code review and the level of detail needed for it	Always	52	3	13.33	12	Very Good	48	2	8.57	25
27	To be willing to learn and improve	Always	50	4	15.24	11	Very Good	49	1	2.86	27

[1] **Usage** frequency is measured on a Likert-type scale: 'Always', 'Most reviews', 'About half reviews', 'Less than half reviews', 'Never', and obtained values are ranked into 10 distinct groups by Scott-Knott ESD algorithm [69].

[2] **Importance** rankings are based on the % of respondents who selected given competency. Participants selected up to 5 most important competencies in the survey.

[3] **Proficiency** level is measured on a Likert-type scale: 'Excellent', 'Very Good', 'Good, Fair', 'Poor', and obtained values are ranked into 7 distinct groups by Scott-Knott ESD algorithm [69].

[4] **Preference to Improve** rankings are based on the % of respondents who selected given competency. Participants selected up to 5 competencies they would like to improve in the survey.



(a) Usage of competencies

(b) Importance of competencies

(c) Proficiency levels for competencies

(d) Competencies preferred for improvement

Fig. 5. Bubble plots showing how experts and novices differ with respect to competencies' usage and importance as well as their proficiency levels and preferences to improve at these competencies. The closer the circles are to the diagonal line, the closer is also the rating of novice and expert reviewers for that competency.

Furthermore, reviewers would like 'to have effective review time-management' (25.71%), 'to give constructive feedback' (23.81%), and 'to estimate the complexity of the problem and solution' (21.90%). More than 20% of reviewers would also like to improve when it comes 'to handling interpersonal conflicts' and 'evaluating the code change quality with respect to its effect on the correctness and evolvability of the software system'.

4.3 Competencies and Experience

In $RQ_{3.1}$ to $RQ_{3.4}$, the competencies of interest are analyzed on the 4 dimensions (frequency of usage, importance, level of proficiency and which ones the reviewers would like to improve) comparing novice against expert reviewers. We used the same approach to process the data as in previous RQs but for separate data sets of novice and expert reviewers. This resulted in separate rankings of competencies for novices and experts as presented in Table 5 and directly compared in Figure 5. The sorted lists of competencies to directly compare novices and experts are presented in Appendix C.

4.3.1 Frequency of Usage. Novice and expert reviewers' rankings of competencies present many similarities. The ranks of competencies on how frequently reviewers need them are very strongly correlated for novice and expert reviewers ($\rho(25) = 0.93$, $p<.001$), as also seen in Figure 5(a). The top ten most frequently used competencies are identical for novice and expert reviewers, with minor differences in the rank. Using the Mann-Whitney U test we have found that expert reviewers need to understand the purpose of the review and the needed level of detail more frequently than novice reviewers (Novice rank = 3, Expert rank = 1). There is a statistical difference in how often these two groups need 'to identify patterns in the code' (Novice rank = 6, Expert rank = 5). While expert reviewers use this competency most frequently in most reviews, novice reviewers need this competency most frequently in less than half reviews.

4.3.2 Importance. Respondents had to select what they consider the most important competencies (up to five) for code review. The importance rank is strongly related between novice end expert reviewers ($\rho(25) = 0.86$, $p<.001$). Figure 5(b) depicts this relationship. In the ten most important competencies, novice and expert reviewers differed the most in how they perceived the competence 'to evaluate the code change quality with respect to the user-affecting properties of the software system'. Expert reviewers find it more important than novices (Novice rank = 12 (selected by 11.36% of novice reviewers), Expert rank = 8 (19.67%)). On the contrary, novice reviewers find it more important than expert reviewers 'to identify patterns in the code' (Novice rank = 9 (20.45%), Expert rank = 12 (13.11%)).

Novice and expert reviewers differ considerably in how important they perceive several other competencies. On the one hand, experts see as more important 'to be willing to compromise' (Novice rank = 21 (4.55%), Expert rank = 12 (13.11%)) and 'to understand the dynamics of the discussion and its conclusions' (Novice rank = 26 (0%), Expert rank = 19 (4.92%)). On the other hand, novice reviewers see it as more important to be able 'to estimate the complexity of the problem and solution' (Novice rank = 10 (15.91%), Expert rank = 18 (6.56%)).

4.3.3 Proficiency level. Novice and expert reviewers also consider themselves proficient in similar competencies. There is a strong correlation of $\rho(25) = 0.76$, $p<.001$) as visualized in Figure 5(c). Using Mann-Whitney U test, we identified five technical competencies and one personal competency in which expert reviewers report a higher level of proficiency than novice reviewers. These are 'to develop a mental model of the change', 'to identify patterns in the code', 'to evaluate the architectural implications of the change, its effect on user-affecting properties and evolvability of the software system', and 'to understand the purpose of the review and the needed level of detail'.

4.3.4 Preference to improve. Novice and expert reviewers want to improve on relatively different sets of competencies: There is a moderate relationship of $\rho(25) = 0.42$, $p<.001$ between their rankings. As seen in Figure 5(d), there is a considerable variety in the rankings of competencies the two groups would like to improve. Table 6 describes the list of competencies up to rank 10 (13 competencies). While novice reviewers would like to improve the most in technical competencies (7 out of the first 13), expert reviewers' preferences are more oriented to improve their social skills (6 out of

the 13). The competencies that both novice and expert reviewers would like to improve are 'to evaluate the architectural implications of the change', 'to communicate in an understandable way', 'to evaluate the code change quality with its respect to correctness and evolvabilty of the software system', 'to have effective review time management', 'to estimate the complexity of the problem and solution' and 'to handle interpersonal conflicts'. Novice reviewers put preference on improving their ability 'to evaluate the code change quality with respect to the user-affecting properties of the software system', 'to set, express and manage priorities', 'to develop a mental model of the change', 'to identify patterns in the code' and 'to effectively manage own emotions'. In comparison, the expert reviewers would like 'to give more constructive feedback', 'to mentor other developers', 'to convince others about their ideas', 'to adjust communication style to different individuals', and 'to be more aware of the capacity of their co-workers in terms of time, experience and knowledge'.

5 DISCUSSION

A number of conclusions can be drawn from the observations about developers' code review competencies from the data we collected to answer the research questions. We describe the implications of these results in Section 5.2. Section 5.3 presents the future work and Section 5.4 present the threats to validity.

5.1 Findings

As many as 27 competencies are needed for code review. While there is a body of knowledge describing the skills and competencies needed for IT professionals, software engineers, and specific software development roles [4], the perspective of providing needed competencies for a specific task, such as code review, was not yet explored in the software engineering literature. Code review plays a crucial role in software quality assurance [2, 5, 44]. It is also used as a training process within software development teams [5] and as a tool in developers' education [3]. This study is the first that identified a list of 27 competencies relevant for code reviews.

Technical competencies are a baseline for efficient code reviews. Among all competencies, technical ones are reported to be the most frequently used and the ones perceived as the most important. They constitute an important baseline to perform efficient and effective code reviews. Developers are also the most confident in their proficiency with respect to these competencies. Even though software tools [21, 48, 70] and reading techniques [8, 58] have been devised to support reviewers, especially in the technical part of the review, we know less about what competence developers need to conduct better reviews beyond identifying more defects or maintainability issues. Our study underlined that, apart from assessing the correctness of the code change or its effect on the evolvability of the software system, it is also important for reviewers to develop a good mental model of the code change, evaluate it against the requirements, and correctly apply programming languages and paradigms. Reviewers also specifically wish to improve in evaluating the architectural implications of the change.

Clear communication and constructive feedback need more attention. Our data provide initial evidence that reviewers are less equipped for the *social* competencies of the review, even though our respondents report that understandable communication and constructive feedback are frequently needed and very important for the quality of code review. This discrepancy is also reflected in the literature. Many studies, methods, and tools are concerned with supporting developers' ability to understand the code change and identify defects [8, 21, 48, 58, 70]. However, the literature that investigates and supports communication during code review is scarce [22, 50, 74]. Therefore, this mismatch provides an indication of what could be an impactful, yet unexplored part of the code

Table 5. Ranking of Competencies for All, Novice and Expert Reviewers

Competency	Usage[1]			Importance[2]			Proficiency[3]			Preference to Improve[4]		
Technical	**All**	**Novice**	**Expert**	**All**	**Novice**	**Expert**	**All**	**Novice**	**Expert**	**All**	**Novice**	**Expert**
Cluster 1: *Change understanding*												
To develop a mental model of the change	3	2	2	1	2	1	3	4	2	10	8	13
Cluster 2: *System and codebase navigation*												
To systematically navigate codebase	7	6	6	14	16	12	2	2	3	22	22	19
To find relevant information beyond the codebase	9	8	9	12	11	12	2	2	3	17	22	13
To identify patterns in the code	6	6	5	9	9	12	3	4	3	15	10	16
To proficiently use the tools used in the team	4	4	3	14	14	16	2	2	3	25	26	23
Cluster 3: *Software quality and its assessment*												
To evaluate if the implementation fulfills the requirements	3	2	2	2	1	2	2	2	2	26	22	26
To evaluate architectural implications of the change	6	6	4	6	6	7	3	5	3	1	1	7
To evaluate the code change quality with respect to its effects on the correctness of the software system	3	3	2	3	3	3	3	3	4	6	3	7
To evaluate the code change quality with respect to its effects on the user-affecting properties of the software system	5	5	4	9	12	8	4	5	4	9	2	19
To evaluate the code change quality with respect to its effects on the evolvability of the software system	4	3	3	7	8	6	3	4	3	7	3	11
To estimate complexity of the problem and solution	6	5	6	14	10	18	3	3	3	5	10	4
To know and correctly use programming languages	3	2	2	8	7	8	1	1	2	21	14	22
Social												
Cluster 4: *Discussing informatics topics*												
To communicate in an understandable way	1	1	1	5	5	3	3	2	4	2	3	1
To give constructive feedback	2	2	1	3	3	5	3	3	4	4	13	2
To understand the dynamics of the discussion and its conclusions	6	6	5	23	26	19	4	4	5	19	21	16
Cluster 5: *Relating to colleagues*												
To adjust communication style to different individuals	7	7	6	23	24	21	6	5	6	14	16	7
To pick up and act on ideas of others	7	6	7	22	21	21	3	3	4	22	16	22
To be willing to compromise	6	6	6	18	21	12	4	3	6	17	16	16
To convince others about your ideas	8	7	8	20	16	21	5	5	6	13	16	6
To mentor other developers	7	6	6	14	14	16	4	4	5	10	20	4
To effectively manage own emotions	8	7	8	20	16	21	5	5	6	16	10	19
To handle interpersonal conflicts	10	9	10	26	24	26	6	6	7	7	8	7
To be aware of the capacity of co-workers in terms of time, experience and knowledge	6	5	5	19	16	19	4	4	5	19	27	11
Cluster 6: **Personal**												
To have effective review time-management	6	5	5	26	26	21	7	7	8	3	3	3
To set, express, and manage priorities	7	5	7	23	21	26	6	6	6	10	3	15
To understand the purpose of the code review and the level of detail needed for it	3	3	1	12	16	10	2	3	2	25	14	25
To be willing to learn and improve	4	3	3	11	12	10	1	1	1	27	22	27

[1] **Usage** frequency rankings are produced by using the Scott-Knott ESD algorithm [69].
[2] **Importance** rankings are produced by using % of developers who selected the competency as the most important.
[3] **Proficiency** level rankings are produced by using the Scott-Knott ESD algorithm [69] .
[4] **Preference to Improve** rankings are produced by using % of developers who selected the competency as preferred for improving in themselves.

review process for future research. Furthermore, code review training and tooling might benefit from explicitly focusing on improving the quality of communication during code reviews.

Novice reviewers would like to improve in understanding and contextualizing the change. The biggest difference between novice and expert reviewers concerns what competencies they would like to improve. Unlike experts, novices would like to improve in developing a mental model of the change (the most important competency) and identifying patterns in the code. In the terms of the SECAT model [62], these competencies relate to the first level of *functional competence*, where students need to clearly process the technical information. This might also reflect that code reviews

Table 6. Comparison of Competencies Novice and Expert Reviewers would Prefer to Improve

Novice	Type[1]	Rank	Expert	Type	Rank
To evaluate architectural implications of the change	T	1	To communicate in an understandable way	S	1
To evaluate the code change quality with respect to its effects on the user-affecting properties of the software system	T	2	To give constructive feedback	S	2
To communicate in an understandable way	S	3	To have effective review time-management	P	3
To evaluate the code change quality with respect to its effects on the correctness of the software system	T	3	To mentor other developers	S	4
To evaluate the code change quality with respect to its effects on the evolvability of the software system	T	3	To estimate complexity of the problem and solution	T	4
To set, express, and manage priorities	P	3	To convince others about your ideas	S	6
To have effective review time-management	P	3	To evaluate the code change quality with respect to its effects on the correctness of the software system	T	7
To develop a mental model of the change	T	8	To evaluate architectural implications of the change	T	7
To handle interpersonal conflicts	S	8	To adjust communication style to different individuals	S	7
To estimate complexity of the problem and solution	T	10	To handle interpersonal conflicts	S	7
To identify patterns in the code	T	10	To evaluate the code chang quality with respect to its effects on the evolvability of the software system	T	11
To effectively manage own emotions	S	10	To be aware of the capacity of co-workers in terms of time, experience and knowledge	P	11

[1] The type of competency is read as: T - technical, S - social, P - personal. The table compares the first 13 competencies (up to rank 10 for novices and 11 for experts.

are often used as part of on-boarding and mentoring to get acquainted with the code base [6, 36]. Next, novices would like to better evaluate the user-affecting properties and effect of the code change. In the SECAT model [62], this competency reflects the *process competence* and the need of novices to contextualize the change with respect to the usability of the software system. Novice reviewers would also like 'to better set, express, and manage priorities' - a competency related to the third level of the SECAT model [62] where the sensitivity to context and efficient organization of own work should be achieved. Finally, code reviews are an activity where interpersonal conflicts are likely to happen [74]. Novices and experts both would like to improve in handling conflicts effectively. However, novices specifically wish to also learn to effectively manage their own emotions–a competency that expert reviewers may have already achieved with experience. In conclusion, even though novice reviewers prefer to improve mostly their technical competencies, the competencies they would like to develop represent the full spectrum of SE competence levels.

Experts want to improve collaboration, mentoring, and convincing others about their ideas. According to literature, the technical competence peaks at the Senior Software Engineer positions, while the more general competencies gain importance at the managerial level [17]. This is also reflected

in our study. Expert reviewers want to improve mostly their social competencies. Unlike novice reviewers, experts would like to improve competencies related to communication and collaboration with others, but also take over responsibilities for mentoring other developers (the third level of the SECAT model [62]). These preferences might also reflect taking over responsibilities for more complex reviews. Code reviews serve as an onboarding process for newcomers [36]. It is common that new team members firstly have their code reviewed, then they also become reviewers and, with time, mentors. Furthermore, code reviews are becoming more lightweight and most of the changes are not very extensive [59]. Therefore, the need to develop more social competencies can be related to the growing complexity of reviews that the developer is performing. The complexity of the reviews is one of the factors affecting the presence of interpersonal conflicts and differences in opinions [74]. More complex reviews might require also deeper and more extensive discussion, thus better social competence to communicate effectively.

No competencies can be disregarded. Among the competencies that respondents report using the least frequently and that they consider the least important, we find convincing others about own ideas, effectively managing own emotions, and handling interpersonal conflicts. However, the survey respondents indicate that these are the competencies in which they lack proficiency or the competencies where they would like to improve. In this light, handling interpersonal conflicts is an interesting case. Most often, reviewers need it in less than half of the reviews (N=54). However, they also report that 27.62% of reviewers need this competency in half or more reviews. Conflicts can indeed be an obstacle in performing efficient code reviews [23, 74]. Therefore, less frequently used competencies may not be a priority when designing a training or tool support, but one should not draw the conclusion that they are irrelevant to performing efficient reviews. Furthermore, having effective review time management was not rated as frequently used or important. However, reviewers mention it as one of the competencies with the lowest level of proficiency and the third most preferred competency to be improved.

5.2 Implications

Competencies and competency modeling have a wide range of applications and usage, the most common ones being recruitment, training, and performance assessment (see also Section 2.3) [38]. Developers, teams, and companies can benefit in all these areas to improve their code review practice. When conducting recruitment and training activities or assessing code review performance, the competency-based perspective can offer fine-grained developer-oriented criteria to design such activities and evaluate developers' performance. For example, developing a mental model of the code change is one of the crucial competencies reviewers identify as needed to perform code reviews well. Developers can be tested on how well, quickly, or accurately they develop mental models of the reviewed change and whether their competence to do so improves over time. Furthermore, this study provided insights into what cognitive processes happen during code review. These insights can be used to improve the tools used for code reviews and hint at what support reviewers need to perform better reviews.

5.2.1 Assessing Code Review Performance. Code review performance is traditionally evaluated by review effectiveness and efficiency [8]; this is a perspective focused on finding defects as the main goal of code review [5]. Even though finding defects is the main goal of performing code reviews, a review that does not identify defects is not necessarily a review that was poorly done given that not all review changes do not contain correctness defects. Alternatively, code reviews can be assessed by how well they fulfill other review goals, such as knowledge sharing [26]. Competency modeling offers yet another perspective, that assesses how well the reviewers performed the review, rather than assessing a specific review outcome. In other words, thanks to identifying and

ranking the needed competencies by importance, a good reviewer is not only a reviewer that can find correctness defects, but also a reviewer that can develop an accurate mental model of the change, who is able to evaluate whether the implementation fits requirements, gives constructive feedback, and communicates understandably. Depending on the context and review, other criteria and competencies might be more important to assess, such as their ability to evaluate how the change affects the software architecture, the user, or the interpersonal conflicts in the team.

5.2.2 Designing Recruitment, Performance Assessment, Training and Education Activities. Code review is an essential practice to improve code review quality that can suffer from lack of skills [3, 41]. This study shows that not only the technical competencies but also the social ones need attention to help students and practitioners perform reviews well. Competency-Based Education and Training [9] stands on the selection of a smaller set of competencies for which a focused training or activity is designed to allow deliberate practice, repetition, and gradual improvement based on individual feedback [25]. This might be achieved by designing curricula activities that enable learners to demonstrate and discuss with peers and educators the strategies they use to understand and contextualize the change, solve technical problems, and give feedback to others.

A similar principle is used in designing Competence-based Assessment Centers in which a group of developers would participate in several such activities [47]. Their performance can be assessed using the defined competencies either for the purpose of hiring or team-based training. Then, the level of their competencies would be assessed before and after the training sessions. For designing these activities, first, a selection of the crucial competencies would be made (*e.g.*, the five most frequently used in code reviews). Then, activities and code reviews for assessment and training would be selected that enable the participants to manifest their competence in a subset of those (Commonly, a set of 3-4 competencies is observed within one activity.). Then, a set of criteria and behavioral anchors would be defined to assess better and worse manifestations of the competence in the particular scenario. Therefore, this study provides a way to select the relevant competencies but leaves a space for the specific training design.

5.2.3 Identifying Training Needs. Training can be designed with different priorities. This study identified 27 competencies relevant for code review performance. If more general training is being designed, the designers can focus on the most frequently used competencies or the more important ones. However, novice and expert reviewers report different needs for developing their reviewing competence. During onboarding or training of novice reviewers, it can be beneficial to focus on their technical competencies, improving their knowledge of the company's software system to support their better understanding of the change and their ability to contextualize it within the code base and its architecture. The experienced reviewers can benefit from training focused more on mentoring, giving feedback, and adjusting communication with different types of colleagues.

What reviewers would like to improve is the area with the least agreement among respondents. It seems reasonable to interpret that the need for developing competencies is highly individual. Similarly to RQ2, the list can be used as a tool to facilitate the identification of training needs of specific reviewers or whole teams by identifying which competencies they struggle with and which they would like to improve. This input can serve to design custom training activities.

5.2.4 Supporting Code Review Performance. The set of competencies and their ranking can also be used to better understand what cognitive processes reviewers follow and what behaviors they need to manifest. Therefore, it is also a list of activities and processes that might benefit from further support from tooling or management to improve the individual technical part of code review, the communication between the author and reviewers, and personal and team coordination of the review process.

Providing better support for the technical competence has been substantially covered by research, *e.g.*, by approaches to help reviewers to understand the code change and identify defects [8, 21, 48, 58, 70]. Visualizations of the code changes or untangling and scaling down changes are useful to help develop an appropriate mental model of the code change. However, reviewers could benefit from support in further areas, such as (1) clearly communicating and expressing requirements related to the change, and providing explicit ways to check their fulfillment, (2) mastering the use of the teams' tools, (3) identifying evolvability and maintainability issues in the code with suggestions or automatic checks, (4) improving proficient application of the language and programming paradigm in use, (5) understanding how the overall software architecture is affected by the code change, (6) having fast access to needed information across the code base (but also in other resources, like language documentation), and (7) improving communication between the author and reviewer about the complexity of the review and code change (because misalignment in the perception and expectation of problem and solution complexity can be a source of conflicts and misunderstanding in reviews [74]). All of these tasks could benefit from direct handling within the code review tools and minimizing the need for context switching when doing reviews.

The social competencies have received much less support in terms of research and tooling than the technical ones [50, 54, 74]. Especially, supporting reviewers in providing constructive feedback is an important area to cover with research and tooling improvements. Constructive feedback in code reviews has been described as feedback explaining reasons for decisions, providing suggestions with supporting examples, being specific on what to do, how to do it, and what is wrong while prioritizing vital issues. Furthermore, it should also include positive points [74]. Reviewers might also benefit from ways (1) to support clear communication about conclusions from the debate, for example by having a direct way to define a follow-up issue or task directly from the review interface, (2) to have a way to express priorities and importance of comments and requests made during the review or (3) to track mentoring and learning goals and progress. Code review tools and also managers might benefit from support in detecting interpersonal conflicts and communication issues during the reviews and having propositions on how to manage the situation [24, 74].

The personal competencies described in this study provide information on the importance of managing the overall review process and helping reviewers and authors have an overview of the (1) purpose of the review, (2) current development and code change priorities, and (3) availability and review effort of their colleagues for reviews, for example by documenting the load of reviews per team member, signaling availability or time-frame for providing feedback.

5.3 Future Work

One of the commonly evaluated performance criteria of reviewers is code review effectiveness and efficiency [8] which relates to the competency of evaluating the effect of the code change on the correctness of the software system. Other studies also investigated reviewers' ability to identify evolvability issues [65]. However, how reviewers perform reviews seems to be related to many more abilities. This study has described a list of competencies that describe such abilities. Competencies are used in practice as a performance assessment criteria [14] and underline the importance of assessing reviewers also on other abilities, such as developing an accurate mental model of the change or whether they are understandable and constructive with their colleagues. We hope this research can serve as a starting point for other researchers to continue competency-based research in this area and to start evaluating the role, manifestation, and effect of different competencies in code reviews and how these performance criteria relate to and support each other in performing reviews well.

This study also supported the importance of social competence for performing reviews. Researchers have invested substantial effort in supporting developers in the technical review (*e.g.*,

understanding the code change and identifying defects [8, 21, 48, 58, 70]). However, the social competencies have received much less support in terms of research and tooling [50, 54, 74]. Future studies can be designed and carried out to investigate these important areas of code review with the goal of supporting them.

While the provided list can be used to select competencies to design training activities, recruitment interviews, and assessment centers, the implementation of such activities still requires operationalizing these competencies into measurable criteria and assessing the validity and reliability of their measurement.

5.4 Threats to validity

Concept Validity. The concept of competency is used in many fields and (applied) contexts. Therefore, competencies are defined in several ways. Studies often do not define what is meant by competency or use the term vaguely [73]. This results in a wide range of workers' characteristics, behaviors, and abilities that are intertwined within one competency model or study. This is also noticeable in the competency models we used to create the initial item pool to define competencies for code reviews. We have addressed this issue during the Refinement and Clustering phase of the Competencies Identification part of this study, described in Section 3.2.3. However, some competencies in our list still carry marks of that (*e.g.*, willingness to improve is a personality trait), but the experts underlined the special importance of this aspect for code reviews, therefore we kept it in the list.

External Validity. We aimed to include as experts in our study professionals with proven success in code reviews, which we translated into the criterion of at least five years of experience with a review. However, the length of experience is not very strongly related to performance for experts [25]. The code review performance might peak after a shorter period of time. However, there is no available data to specify this. Furthermore, qualitative criteria are commonly used to identify experts [40]. These qualitative criteria are not readily available to identify code review experts, even more so criteria that could be applied to identify experts in a survey. Changing this criterion could affect results and preferences in the analysis of the difference between novice and expert reviewers.

The respondents of the survey have been recruited through a multitude of channels to ensure the diversity of reviewer profiles and code review practices. However, we have tracked the origin of the respondents only when collaborating with companies. Therefore, 14% of our sample consists of reviewers from 2 European-based companies. The rest of the reviewers have been recruited from the other resources mentioned in Section 3.5.1 with a non-negligible chance of being biased by recruiting through personal networks of the authors. Furthermore, the survey participants were rather senior: more than half had more than five years of experience with code reviews and only 16 respondents had less than three years of experience. Therefore, also our results are more representative of more senior reviewers and less so of very novice reviewers or students who might need more support performing reviews well.

Conclusion Validity. This study aimed to identify competencies essential for code reviews. For clarity, we have focused on the competencies needed by reviewers. However, in code review also the change author plays a role. The authors might have a different perspective on what competencies are important or should be improved for better reviews. Furthermore, the code authors might also need specific competencies that would make the reviews more efficient, such as 'to write a clear description of the change' and 'limiting the size of the patch'. However, most of the reviewers also submit code changes to be reviewed by other developers; hence act in the role of the code authors in code reviews.

We have been asking reviewers about potentially personal issues, such as their ability to handle conflicts, manage their emotions, or convince others about their ideas. These competencies came in the results at the end of the most used competencies and also the least important. There might be a bias caused by reviewers being uncomfortable with sharing personal information or putting more emphasis on technical competence in general. We aimed to minimize the potential bias by making the survey anonymous, collecting no personal identifiers, and providing no personal gains or losses for the participants. Therefore, we expect that the survey itself did not affect strongly participants' responses and we expect that related personality traits like the comfort to discuss more emotional or personal issues are randomly distributed among the respondents.

The self-reported frequency of using a competency can be distorted by memory biases. Therefore, we aimed not to use a strictly quantifying scale, like 'once in 2 reviews' but rather a generalized statement like 'in most reviews' which is less prone to this group of biases and compares the experience to the overall portion of reviews the reviewer performs.

We have also relied on a self-report of the reviewers' competence. While this might provide bias in the responses, our goal was not to assess the reviewers' actual level of competence but to prioritize which competencies developers perceive as more or less developed. This lowers the bias present in the data. However, it would still be beneficial to actually measure the developers' competence to triangulate the data if the methods to do so are available.

6 CONCLUSION

This study describes 27 competencies relevant to performing code reviews. The results reveal that technical competence is the base requirement for code reviews and reviewers feel confident about them. However, reviewers are less confident in their social competence and would also like to improve it.

Importantly, current research focuses mainly on supporting developers' technical competence for code reviews, even though developers have more issues with social competencies.

The list of competencies and their ranking can be used as an alternative way to assess code review performance, design and execute recruitment or training activities, and identify what support reviewers need to perform code reviews well.

ACKNOWLEDGMENTS

P. Wurzel Gonçalves and A. Bacchelli gratefully acknowledge the support of the Swiss National Science Foundation through the SNF Project No. PP00P2_170529.

REFERENCES

[1] Alain Abran, James W Moore, Pierre Bourque, Robert Dupuis, and L Tripp. 2004. Software engineering body of knowledge. *IEEE Computer Society, Angela Burgess* (2004).

[2] A.F. Ackerman, L.S. Buchwald, and F.H. Lewski. 1989. Software inspections: an effective verification process. *IEEE Software* 6, 3 (1989), 31–36. https://doi.org/10.1109/52.28121

[3] Hiyam Al-Kilidar, Tor Stalhane, Cat Kutay, and Ross Jeffery. 2003. Teaching the Process of Code Review. (2003).

[4] Nana Assyne, Hadi Ghanbari, and Mirja Pulkkinen. 2021. The state of research on software engineering competencies: A systematic mapping study. *Journal of Systems and Software* (2021), 111183.

[5] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 712–721.

[6] Sogol Balali, Igor Steinmacher, Umayal Annamalai, Anita Sarma, and Marco Aurelio Gerosa. 2018. Newcomers' barriers... is that all? an analysis of mentors' and newcomers' barriers in OSS projects. *Computer Supported Cooperative Work (CSCW)* 27, 3 (2018), 679–714.

[7] Dave Bartram. 2006. The SHL universal competency framework. *SHL White paper* (2006), 1–8.

[8] Tobias Baum. 2019. *Cognitive-support code review tools: improved efficiency of change-based code review by guiding and assisting reviewers.* Ph. D. Dissertation. Hannover: Institutionelles Repositorium der Universität Hannover. https:

//doi.org/10.15488/9164
[9] Harm Biemans, Renate Wesselink, Judith Gulikers, Sanne Schaafsma, Jos Verstegen, and Martin Mulder. 2009. Towards competence-based VET: dealing with the pitfalls. *Journal of Vocational Education and training* 61, 3 (2009), 267–286.
[10] Linda Birt, Suzanne Scott, Debbie Cavers, Christine Campbell, and Fiona Walter. 2016. Member checking: a tool to enhance trustworthiness or merely a nod to validation? *Qualitative health research* 26, 13 (2016), 1802–1811.
[11] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of useful code reviews: An empirical study at microsoft. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 146–156.
[12] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2019. An empirical assessment of best-answer prediction models in technical Q&A sites. *Empirical Software Engineering* 24, 2 (2019), 854–901.
[13] Gemma Catolino and Filomena Ferrucci. 2019. An extensive evaluation of ensemble techniques for software change prediction. *Journal of Software: Evolution and Process* 31, 9 (2019), e2156.
[14] Hsin-Chih Chen and Sharon S Naquin. 2006. An integrative model of competency development, training design, assessment center, and multi-rater assessment. *Advances in Developing Human Resources* 8, 2 (2006), 265–282.
[15] Richard E Clark and Fred Estes. 1996. Cognitive task analysis for training. *International Journal of Educational Research* 25, 5 (1996), 403–417.
[16] Jason Cohen. 2010. Modern Code Review. In *Making Software*, Andy Oram and Greg Wilson (Eds.). O'Reilly, Chapter 18, 329–338.
[17] Ricardo Colomo-Palacios, Cristina Casado-Lumbreras, Pedro Soto-Acosta, Francisco J García-Peñalvo, and Edmundo Tovar-Caro. 2013. Competence gaps in software personnel: A multi-organizational study. *Computers in Human Behavior* 29, 2 (2013), 456–461.
[18] Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. 2021. Do you really code? Designing and Evaluating Screening Questions for Online Surveys with Programmers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 537–548.
[19] Nicole Davila and Ingrid Nunes. 2021. A systematic literature review and taxonomy of modern code review. *Journal of Systems and Software* (2021), 110951.
[20] Tapajit Dey, Andrey Karnauch, and Audris Mockus. 2021. Representation of developer expertise in open source software. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 995–1007.
[21] Martín Dias, Alberto Bacchelli, Georgios Gousios, Damien Cassou, and Stéphane Ducasse. 2015. Untangling fine-grained code changes. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 341–350.
[22] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. 2021. An exploratory study on confusion in code reviews. *Empirical Software Engineering* 26, 1 (2021), 1–48.
[23] Carolyn D Egelman, Emerson Murphy-Hill, Elizabeth Kammer, Margaret Morrow Hodges, Collin Green, Ciera Jaspan, and James Lin. 2020. Predicting developers' negative feelings about code review. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 174–185.
[24] Carolyn D Egelman, Emerson Murphy-Hill, Elizabeth Kammer, Maggie Morrow Hodges, Collin Green, Ciera Jaspan, and James Lin. 2020. Pushbacacterizing and Detecting Negative Interpersonal Interactions in Code Review. (2020).
[25] K Anders Ericsson and Andreas C Lehmann. 1996. Expert and exceptional performance: Evidence of maximal adaptation to task constraints. *Annual review of psychology* 47, 1 (1996), 273–305.
[26] Nargis Fatima, Sumaira Nazir, and Suriayati Chuprat. 2019. Understanding the impact of feedback on knowledge sharing in modern code review. In *2019 IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*. IEEE, 1–5.
[27] National Center for O*NET Development. 2020. *O*Net Resource Center*. Retrieved October, 2020 from https://www.onetcenter.org/
[28] Mina Ghomi and Christine Redecker. 2019. Digital Competence of Educators (DigCompEdu): Development and Evaluation of a Self-assessment Instrument for Teachers' Digital Competence.. In *CSEDU (1)*. 541–548.
[29] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*. 345–355.
[30] Joan S Grant and Linda L Davis. 1997. Selection and use of content experts for instrument development. *Research in nursing & health* 20, 3 (1997), 269–274.
[31] Fergus Henderson. 2017. Software engineering at Google. *arXiv preprint arXiv:1702.01715* (2017).
[32] Barbara A Kitchenham and Shari L Pfleeger. 2008. Personal opinion surveys. In *Guide to advanced empirical software engineering*. Springer, 63–92.
[33] Barbara A Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on software engineering* 28, 8 (2002), 721–734.

[34] Ruth Klendauer, Marina Berkovich, Richard Gelvin, Jan Marco Leimeister, and Helmut Krcmar. 2012. Towards a competency model for requirements analysts. *Information Systems Journal* 22, 6 (2012), 475–503.
[35] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W Godfrey. 2015. Investigating code review quality: Do people and participation matter?. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 111–120.
[36] Vladimir Kovalenko and Alberto Bacchelli. 2018. Code review for newcomers: is it different?. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*. 29–32.
[37] Klaus Krippendorff. 2018. *Content analysis: An introduction to its methodology*. Sage publications.
[38] Rainer Kurz and Dave Bartram. 2002. Competency and individual performance: Modelling the world of work. *Organizational effectiveness: The role of psychology* 227 (2002), 255.
[39] Choong Kwon Lee and Stephen C Wingreen. 2010. Transferability of knowledge, skills, and abilities along IT career paths: An agency theory perspective. *Journal of Organizational Computing and Electronic Commerce* 20, 1 (2010), 23–44.
[40] Ming Li and Carol S Smidts. 2003. A ranking of software engineering measures based on expert opinion. *IEEE Transactions on Software engineering* 29, 9 (2003), 811–824.
[41] Xiaosong Li. 2007. Incorporating a code review process into the assessment. (2007).
[42] Barbara Linck, Laura Ohrndorf, Sigrid Schubert, Peer Stechert, Johannes Magenheim, Wofgang Nelles, Jonas Neugebauer, and Niclas Schaper. 2013. Competence model for informatics modelling and system comprehension. In *2013 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 85–93.
[43] Ernest J McCormick, PR Jeanneret, and S Gael. 1988. Position analysis questionnaire (PAQ). In *SYMPOSIUM PROCEEDINGS OCCUPATIONAL RESEARCH AND THE NAVY-PROSPECTUS 1980*. ERIC, 78.
[44] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.
[45] Ana M Moreno, Maria-Isabel Sanchez-Segura, Fuensanta Medina-Dominguez, and Laura Carvajal. 2012. Balancing software engineering education and industrial needs. *Journal of systems and software* 85, 7 (2012), 1607–1620.
[46] Frederick P Morgeson, Adela S Garza, and Michael A Campion. 2013. Work design. (2013).
[47] International Task Force on Assessment Center Guidelines. 2009. Guidelines and Ethical Considerations for Assessment Center Operations 1. *International Journal of Selection and Assessment* 17, 3 (2009), 243–253.
[48] Sebastiaan Oosterwaal, Arie van Deursen, Roberta Coelho, Anand Ashok Sawant, and Alberto Bacchelli. 2016. Visualizing code and coverage changes for code review. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 1038–1041.
[49] Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. 2018. Information Needs in Contemporary Code Review. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 135 (nov 2018), 27 pages. https://doi.org/10.1145/3274404
[50] Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. 2018. Information needs in contemporary code review. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–27.
[51] Thomas V Perneger. 1998. What's wrong with Bonferroni adjustments. *Bmj* 316, 7139 (1998), 1236–1238.
[52] Denise F Polit and Cheryl Tatano Beck. 2006. The content validity index: are you sure you know what's being reported? Critique and recommendations. *Research in nursing & health* 29, 5 (2006), 489–497.
[53] Mohammad Masudur Rahman and Chanchal K Roy. 2014. An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. 364–367.
[54] Mohammad Masudur Rahman, Chanchal K Roy, and Raula G Kula. 2017. Predicting usefulness of code review comments using textual features and developer experience. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 215–226.
[55] Louis M Rea and Richard A Parker. 2014. *Designing and conducting survey research: A comprehensive guide*. John Wiley & Sons.
[56] Peter C Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 202–212.
[57] José Gamaliel Rivera-Ibarra, Josefina Rodríguez-Jacobo, José Alberto Fernández-Zepeda, and Miguel Angel Serrano-Vargas. 2010. Competency framework for software engineers. In *2010 23rd IEEE Conference on Software Engineering Education and Training*. IEEE, 33–40.
[58] Guoping Rong, Jingyi Li, Mingjuan Xie, and Tao Zheng. 2012. The effect of checklist in code review for inexperienced students: An empirical study. In *2012 IEEE 25th Conference on Software Engineering Education and Training*. IEEE, 120–124.
[59] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. 181–190.

[60] J Saldaña-Ramos, Ana Sanz-Esteban, J García-Guzmán, and A Amescua. 2012. Design of a competence model for testing teams. *IET Software* 6, 5 (2012), 405–415.
[61] Neal W Schmitt, Scott Ed Highhouse, and Irving B Weiner. 2013. *Handbook of psychology: Industrial and organizational psychology, Vol. 12.* John Wiley & Sons Inc.
[62] Yvonne Sedelmaier and Dieter Landes. 2014. A multi-perspective framework for evaluating software engineering education by assessing students' competencies: SECAT—A software engineering competency assessment tool. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. IEEE, 1–8.
[63] Yvonne Sedelmaier and Dieter Landes. 2014. Software engineering body of skills (SWEBOS). In *2014 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 395–401.
[64] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Acceptance factors of pull requests in open-source projects. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 1541–1546.
[65] Davide Spadini, Fabio Palomba, Tobias Baum, Stefan Hanenberg, Magiel Bruntink, and Alberto Bacchelli. 2019. Test-driven code review: an empirical study. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1061–1072.
[66] Saikrishna Sripada, Y Raghu Reddy, and Ashish Sureka. 2015. In support of peer code review and inspection in an undergraduate software engineering course. In *2015 IEEE 28th Conference on Software Engineering Education and Training*. IEEE, 3–6.
[67] Claude M. Steele and Jay Aronson. 1995. Stereotype threat and the intellectual test performance of African Americans. *Journal of personality and social psychology* 69 5 (1995), 797–811.
[68] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering* 43, 1 (2016), 1–18.
[69] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2019. The Impact of Automated Parameter Optimization for Defect Prediction Models. *IEEE Trans. Software Eng.* 45, 7 (2019), 683–711.
[70] Yida Tao, Yingnong Dang, Tao Xie, Dongmei Zhang, and Sunghun Kim. 2012. How do software engineers understand code changes? An exploratory study in industry. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. 1–11.
[71] Richard T Turley and James M Bieman. 1994. Identifying Essential Competencies of Software Engineers.. In *ACM Conference on computer science*, Vol. 10.
[72] June Wei and Gavriel Salvendy. 2004. The cognitive task analysis methods for job and task design: Review and reappraisal. *Behaviour & Information Technology* 23, 4 (2004), 273–299.
[73] Charles Woodruffe. 1993. What is meant by a competency? *Leadership & organization development journal* (1993).
[74] Pavlína Wurzel Gonçalves, Gul Çalikli, and Alberto Bacchelli. 2022. Interpersonal Conflicts During Code Review: Developers' Experience and Practices. *Proceedings of the ACM on Human-Computer Interaction* CSCW (2022). https://doi.org/10.5281/zenodo.5851633
[75] FARIDEH YAGHMAEI. 2003. Content validity and its estimation. (2003).

A LIST AND DEFINITIONS OF ALL COMPETENCIES FOR CODE REVIEW

Technical Competencies

Cluster 1 - Change Understanding

(1) To develop a mental model of the change
Developing a mental model of the change and how it relates to the requirements and application domain.

Cluster 2 - System and Codebase Navigation

(2) To systematically navigate the codebase
Systematically exploring the codebase, applying correctly in the change the functions and libraries used in the system.

(3) To find relevant information beyond the codebase
in the documentation, requirements, online resources, literature, etc.

(4) To identify patterns in the code
Identifying patterns in the code and its structural properties, evaluating design patterns implementation, identifying what parts of the code are or can be reused.

(5) To proficiently use the tools used in the team
Code review tools, IDEs, versioning software, issue management systems, etc.

Cluster 3 - Software Quality and Its Assessment

(6) To evaluate if the implementation fulfills the requirements

(7) To evaluate architectural implications of the change

(8) To evaluate the code change quality with respect to its effects on the correctness of the software system

(9) To evaluate the code change quality with respect to its effects on the user-affecting properties of the software system
such as usability, performance, and security.

(10) To evaluate the code change quality with respect to its effects on the evolvability of the software system
Such as structuredness, maintainability, and compatibility.

(11) To estimate complexity of the problem and solution
Distinguishing complex and simple problems, estimating relevant complexity measures.

(12) To know and correctly use programming languages
Effectively using programming languages applied in the system with respect to the related programming paradigms (object-oriented programming, functional, procedural programming, etc.)

Social Competencies

Cluster 4 - Discussing Informatics Topics

(13) To communicate in an understandable way

(14) To give constructive feedback
Able and willing to criticize constructively, ask important and relevant questions, suggest adjustments and alternative solutions, helping others realize new ideas.

(15) To understand the dynamics of the discussion and its conclusions
Being able to follow how a discussion is developing over time, how the participants react to each other, and what conclusions can be taken out of the discussion.

Cluster 5 - Relating to Colleagues

(16) To adjust communication style to different individuals

(17) To pick up and act on ideas of others

(18) To be willing to compromise

(19) To convince others about your ideas

(20) To mentor other developers
Guiding less experienced developers to develop their code reviewing and programming skills and their understanding of the team/project/company processes.
(21) To effectively manage own emotions
Dealing with negative emotions and distressing circumstances, accepting negative feedback, dealing with external stress.
(22) To handle interpersonal conflicts
Enhancing productive outcomes of a conflict while minimizing its escalation or harm done.
(23) To be aware of the capacity of co-workers in terms of time, experience, and knowledge

Personal Competencies

(24) To have effective review time-management
Assigning appropriate time resources for the reviews, balance review response time with other responsibilities.
(25) To set, express, and manage priorities
related to software requirements, release planning, bug severity, product backlog in agile development, etc.
(26) To understand the purpose of the code review and the level of detail needed for it
(27) To be willing to learn and improve
Being open to new ideas, able to learn from past mistakes for future reviews.

B LIST OF ALL COMPETENCIES FOR ALL DEVELOPERS SORTED BY RANKS

Frequency of Usage			Importance			Level of Proficiency			Wanting to Improve		
ID*	Type	Rank	ID	Type	Rank	ID	Type	Rank	ID	Type	Rank
13	S	1	1	T	1	12	T	1	7	T	1
14	S	2	6	T	2	27	P	1	13	S	2
8	T	3	8	T	3	26	P	2	24	P	3
1	T	3	14	S	3	6	T	2	14	S	4
12	T	3	13	S	5	5	T	2	11	T	5
26	P	3	7	T	6	2	T	2	8	T	6
6	T	3	10	T	7	3	T	2	10	T	7
10	T	4	12	T	8	13	S	3	22	S	7
5	T	4	4	T	9	14	S	3	9	T	9
27	P	4	9	T	9	8	T	3	1	T	10
9	T	5	27	P	11	1	T	3	20	S	10
7	T	6	3	T	12	10	T	3	25	P	10
24	P	6	26	P	12	7	T	3	19	S	13
11	T	6	2	T	14	11	T	3	16	S	14
4	T	6	5	T	14	4	T	3	4	T	15
18	S	6	11	T	14	17	S	3	21	S	16
15	S	6	20	S	14	9	T	4	3	T	17
23	S	6	18	S	18	18	S	4	18	S	17
20	S	7	23	S	19	15	S	4	23	S	19
25	P	7	19	S	20	23	S	4	15	S	19
16	S	7	21	S	20	20	S	4	12	T	21
2	T	7	17	S	22	19	S	5	2	T	22
17	S	7	15	S	23	21	S	5	17	S	22
19	S	8	16	S	23	25	P	6	5	T	25
21	S	8	25	P	23	16	S	6	26	P	25
3	T	9	22	S	26	22	S	6	6	T	26
22	S	10	24	P	26	24	P	7	27	P	27

(*) ID of the competency, as specified in Table 4 and Appendix A.

C LISTS OF COMPETENCIES AS RANKED BY NOVICE AND EXPERT REVIEWERS

Frequency of Usage				Importance				Level of Proficiency				Wanting to improve			
Expert		Novice		Expert		Novice		Expert		Novice		Expert		Novice	
ID*	Rank	ID	Rank	ID	Rank	ID	Rank	ID	Rank	ID	Rank	ID	Rank	ID	Rank
1	13	1	13	1	1	1	6	1	27	1	12	1	13	1	7
1	14	2	12	2	6	2	1	2	26	1	27	2	14	2	9
1	26	2	6	3	13	3	14	2	1	2	6	3	24	3	13
2	8	2	14	3	8	3	8	2	12	2	13	4	20	3	8
2	1	2	1	5	14	5	13	2	6	2	3	4	11	3	10
2	12	3	27	6	10	6	7	3	10	2	5	6	19	3	25
2	6	3	26	7	7	7	12	3	5	2	2	7	8	3	24
3	10	3	8	8	12	8	10	3	7	3	14	7	7	8	1
3	5	3	10	8	9	9	4	3	4	3	8	7	16	8	22
3	27	4	5	10	27	10	11	3	11	3	11	7	22	10	11
4	7	5	11	10	26	11	3	3	2	3	26	11	10	10	4
4	9	5	9	12	2	12	27	3	3	3	17	11	23	10	21
5	24	5	23	12	3	12	9	4	13	3	18	13	1	13	14
5	23	5	25	12	4	14	5	4	14	4	1	13	3	14	12
5	4	5	24	12	18	14	20	4	8	4	10	15	25	14	26
5	15	6	2	16	5	16	26	4	9	4	4	16	4	16	17
6	20	6	7	16	20	16	23	4	17	4	20	16	18	16	18
6	11	6	4	18	11	16	2	5	23	4	23	16	15	16	19
6	16	6	17	19	15	16	19	5	15	4	15	19	9	16	16
6	18	6	18	19	23	16	21	5	20	5	7	19	2	20	20
6	2	6	15	21	17	21	25	6	16	5	9	19	21	21	15
7	25	6	20	21	19	21	17	6	18	5	19	22	12	22	27
7	17	7	19	21	21	21	18	6	25	5	21	22	17	22	6
8	19	7	21	21	16	24	16	6	19	5	16	23	5	22	3
8	21	7	16	21	24	24	22	6	21	6	25	25	26	22	2
9	3	8	3	26	25	26	24	7	22	6	22	26	6	26	5
10	22	9	22	26	22	26	15	8	24	7	24	27	27	27	23

(*) ID of the competency, as specified in Table 4 and Appendix A.

Received January 2022; revised July 2022; accepted November 2022