Envisioning software engineer training needs in the digital era through the SWEBOK V4 prism

Hironori Washizaki[1,*], Maria-Isabel Sanchez-Segura[2,*], Juan Garbajosa[3], Steve Tockey[4], Kenneth E Nidiffer[5]
[1]*Waseda University*, Tokyo, Japan washizaki@waseda.jp
[2]*Universidad Carlos III de Madrid*, Madrid, Spain misanche@inf.uc3m.es
[3]*Universidad Politécnica de Madrid*, Madrid, Spain juan.garbajosa@upm.es
[4]*Construx Software*, Bellevue, WA, United States steve.tockey@construx.com
[5]*George Mason University*, Fairfax, VA, United States knidiffe@gmu.edu

***Abstract*—Our world's needs have evolved dramatically since the origins of software engineering in the 1960s. The future software engineer must be able to anticipate our needs and desires in an era where complex challenges continually emerge, and adaptive solutions must be delivered on the fly. This paper addresses the evolution of the IEEE Computer Society's Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) and its impact on software engineering higher education and professional training that should prepare engineers to fulfill their mission in this dynamic digital future.**

***Index Terms*—Education, professional training, SWEBOK**

I. Introduction

This is the world that software engineers must be able to address to anticipate the needs and desires of an era of emerging and complex challenges where solutions will have to be delivered on the fly. The question is, are we training software engineering professionals capable of dealing with the demands of the digital era in the coming years?

In this paper, we outline how and why the new IEEE Computer Society's Guide to the Software Engineering Body of Knowledge (hereafter "SWEBOK V4") [1] takes into consideration the needs of digital era software engineers immersed in an industry demanding visionary solutions and proposes how to train these engineers in the skills that they need to fulfill their mission. In the following, we first present SWEBOK V4 and the evolution point of each Knowledge area. And then, we give the impact of each knowledge area on higher education and training through the prism of SWEBOK V4.

II. SWEBOK V4 at a glance

SWEBOK represents the current state of generally accepted, consensus-based knowledge emanating from the interplay between software engineering theory and practice [2], [3]. Its objectives include the provision of guidance for learners, researchers, and practitioners to identify and share a common understanding of "generally accepted knowledge" in software engineering, defining the boundary between software engineering and related disciplines, and providing a foundation for certifications and educational curricula.

The origins of SWEBOK go back to the early 2000s. Much like the software engineering discipline, SWEBOK has

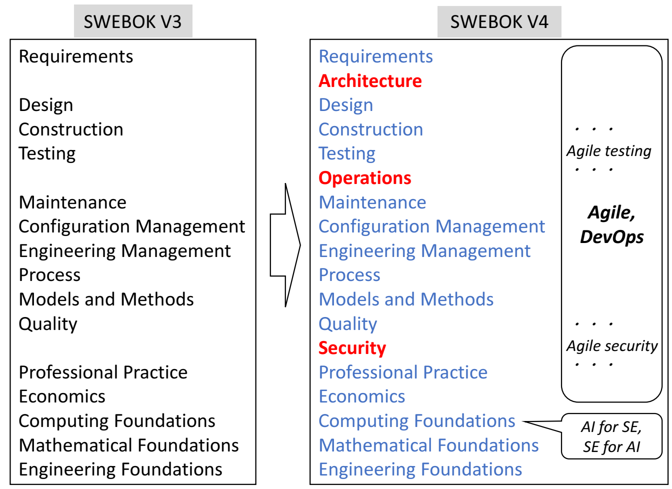

Fig. 1. From SWEBOK V3 to SWEBOK V4

continued to evolve over the last 20 years to reflect the educational, industrial, social, technical, and technological changes in society. A significantly updated SWEBOK V4 will be released in 2023 to improve the guide's currency, readability, consistency, and usability. The current draft public version of SWEBOK V4 can be reached at [1]. The guide contains 18 knowledge areas (KAs), followed by several appendices. A KA is an identified area of software engineering defined by its knowledge requirements and described in terms of its concepts, component processes, practices, inputs, outputs, tools, and techniques. SWEBOK has been adopted well as a fundamental basis for continuously developing software engineering curricula and industrial training programs. For example, [4] proposed a process for assessing and enhancing curricula against SWEBOK and reported the result of a case study targeting multiple universities.

Figure 1 summarizes the transition from SWEBOK V3, published in 2014, to V4. KAs have been updated to reflect the broad acceptance of agile methodologies and DevOps in many differently sized software-intensive organizations and acknowledge the rise of artificial intelligence (AI). Tables I, II, and III collect the main factor of the evolution of each of the SWEBOK KAs. There are three new KAs: Software Architecture, Software Operations, and Software Security. These additions better reflect contemporary software engineering practice, the pressing need to address cybersecurity as early

* Corresponding authors

TABLE I

SWEBOK EVOLUTION AND IMPACTS ON HIGHER EDUCATION AND TRAINING: KA1–KA6

KA	Point of evolution	Impact on higher education	Impact on professional training
KA1. Software Requirements	The importance of requirements documentation for long-term maintenance is added. The whats and hows of work on software requirements in a project should be determined by the type of constructed software and not by the project life cycle. And also it included a deeper, broader coverage of requirements specification techniques, including model-driven requirements specification and acceptance criteria-based requirements specification.	Students should be confronted with a variety of realistic cases where they can experience and practice diverse (e.g., acceptance criteria-based and model-driven) requirements techniques in various project-based situations that address not only new development but also the role of appropriate requirements in ongoing software maintenance.	Practicing professionals have the advantage of real-world experience where the critical requirements knowledge and skills can possibly be applied retroactively to project situations that they have already worked on.
KA2. Software Architecture	As software tends toward larger, evermore complex systems, architecture concerns move beyond construction to connectivity while assuring new levels of quality for safety, security, and dependability. Therefore, it is now a new KA. Software architecture aims to "satisfice" all stakeholders, while the focus of software design is on the transformation of that vision into a feasible implementation.	As software engineering technologies become increasingly powerful while maturing at record speed, things formerly taught abstractly can now be taught practically: students will have to deal with larger systems than ever before and need exposure to the underlying fundamental principles and practices of the discipline.	Architecture has increased in importance as today's cyber-physical systems are pervasive. At the same time, demands for agile, just-in-time solutions accelerate the need to "get things right" early and often. Practitioners, therefore, depend upon knowledge of fundamental principles to survive.
KA3. Software Design	Software is pervasive and critical in all aspects of modern life. The success of pervasive software depends on broadening the background of people with design skills, especially as emerging technologies, faster delivery times, and emphasis on agile, lean, and incremental design impact the development process.	In an era of agile, DevOps, cloud, and other innovations, it is worth remembering that despite huge changes, design is governed by basic principles that apply across the board.	Industry is inhabited by many senior programmers promoted to designers. There is a need for a sound curriculum to ensure practitioners are comfortable with all the key aspects of software design.
KA4. Software Construction	The following sections were added to reflect modern construction techniques and practices: managing dependencies, cross-platform development and migration, feedback loop for construction, and visual programming and low-code/zero-code platforms. Furthermore, major updates were made to some of the existing sections, especially about reuse, life-cycle models, construction languages and environments.	Students must be aware that software construction is not just programming functioning code but creating and maintaining readable and verifiable code through coding, verification, testing, and debugging with consideration of related areas such as Software Design KA. Students should learn and practice construction basics through coding tests and good examples, as well as construction fundamentals, including complexity, change, reuse, coding standards, and the proper support of tools and platforms with a particular focus on modern development processes such as Agile and DevOps.	In addition to construction fundamentals, practitioners need to strengthen the knowledge and capability of cloud-based software development and deployment, the understanding of DevOps and continuous integration/delivery, and the awareness of complex open-source and commercial software ecosystems and their impact on software development in training. Furthermore, practitioners should understand industrial practices in construction management and the spread of reliable construction technologies (such as error handling) as well as modern tools (such as low-code/zero-code platforms).
KA5. Software Testing	The chapter structure has been revised to align with the shift left testing movement. In particular, this KA provides new content about software testing in recent development processes (like Agile, DevOps, and Test-Driven Development), application domains (like automotive, healthcare, mobile, legal, and IoT), emerging technologies (such as AI, blockchain, cloud) and quality attributes (like security and privacy).	It is essential to address the followings: (1) Promote knowledge about quality and its value to raise awareness about the product and process trustworthiness by incorporating specific testing aspects such as automation, machine learning, cloud computing, DevOps, continuous testing, and security into the curriculum. (2) Round out student knowledge with hands-on experiences in laboratory and industrial contexts to raise appreciation of the importance of quality and promote awareness of product and process trustworthiness. (3) Incorporate into the curriculum specific testing aspects such as automation, machine learning, cloud computing, DevOps, continuous testing, and security. (4) Increase student knowledge and skills about emerging software testing technologies and tools.	This KA increases professional knowledge about the risks and consequences of poor software testing. It promotes learning programs and specific courses for addressing critical aspects of software testing and knowledge about emerging testing technologies and tools. It promotes learning-by-doing practices and experience-based learning to improve the awareness of testing and the connected risks.
KA6. Software Engineering Operations	This is a new knowledge area that addresses the evolution of the role of software engineers to include DevOps and infrastructure as code activities while eliminating the organizational silos between development, maintenance, and operations. This new KA describes operations fundamentals, planning, delivery and control activities, and techniques. It also presents practical considerations and tools.	Students must be aware of the importance of software engineering operations, including deployment, configuration, operational monitoring, and management in the era of continuous software engineering. Students should learn fundamentals and practice operation basics and tools, particularly DevOps practices, through exercise and project-based learning.	In addition to operations fundamentals key practices, practitioners should grasp the entire picture of operations, including operation planning, delivery, and control as an integral part of system and software life cycle processes. Furthermore, practitioners should understand modern infrastructure practices such as Agile Infrastructure and Infrastructure as Code (IaC), as well as practical considerations, related standards, and tools such as containers and virtualization.

TABLE II
SWEBOK EVOLUTION AND IMPACTS ON HIGHER EDUCATION AND TRAINING: KA7–KA11

KA	Point of evolution	Impact on higher education	Impact on professional training
KA7. Software Maintenance	Maintenance categories and the software process have been updated to align with the 2021 version of ISO14764. Also, continuous integration, delivery, testing, and deployment have been added as a new topic in response to the growing popularity of regrouping development, maintenance, and operations tasks to improve software engineering productivity.	Maintenance has increased in cost and importance in a world of continuous changes. Although it is often hard to experience the reality of maintenance in higher education programs, students should learn maintenance fundamentals by referring to textbooks, industrial maintenance project reports as well as open-source projects. Furthermore, software architecting and design exercises can teach students how to handle maintainability.	In addition to software maintenance fundamentals, practitioners can further learn maintenance activities, processes, techniques, and tools in Agile and DevOps environments where development, maintenance, and operations are grouped together. Learners are also expected to increase their awareness about the need for maintenance as well as their associated challenges in practice.
KA8. Software Configuration Management	Version control is the best-known facet of the SCM process, although it is just one of many. This helps explain past precariousness of the process. In SWEBOK V4, SCM has been explained considering all of its facets. It stresses that the SCM process plan must be defined first before all the decisions and commitments included in the plan are somehow incorporated into existing tools for execution rather than the other way around, as was the usual practice in the past. Keeping track of CI relationships is essential to visualize the potential impact of a change on a CI over other CIs.	This process cannot be explained without project management (estimation and planning), quality and construction, because it has to be developed in combination with all these processes to guarantee the control of the configuration items to be developed. Of the software engineering teaching methods identified by [5], there is no doubt that the SCM process must be taught using project-based and active learning. Both approaches force students to address SCM integrally, because, for instance, no configuration item can be output unless the project is under development, and no output product can be assessed from the quality assurance process perspective unless it has been identified and cataloged as a configuration item with the identification of its respective baseline.	The configuration plan definition and development have to be strongly connected to project planning and DevOps performance.
KA9. Software Engineering Management	The scale of complicated and complex software-enabled systems and services will continue to increase exponentially with intricate and often hidden interfaces and interrelationships operating in a dynamic and nondeterministic world. As a result, software engineering management knowledge will evolve continuously to meet the needs of society. Although significant progress has been made to date, artificial intelligence/machine learning and other technological systems are beginning to emerge. They will unleash future challenges for systems management.	Modern software engineering management practices are changing to meet the needs of society caused by increases in the size and complexity of software, alongside greater pressure to quickly release software enabling products to market in rapidly changing environments. There are fundamental management shifts in how work is performed because software has the power to cause increasing globalization, innovations, interactions, and productivity and the countervailing force of increasing complexity. These management challenges need to be addressed to produce and sustain future software-enabled systems.	To meet these challenges, a higher education program in software engineering management needs to offer a unique combination of advanced technical knowledge and management competencies. There is a need to develop perspectives for (1) Understanding industrial practices and current and future trends in technology development, (2) Judging and improving software quality, methods, processes, and tools, (3) sustaining software systems over long time periods, (4) innovating software development practices and improving performance, and (5) a project-based, collaborative learning environment to meet the needs of the workforce in the digital age.
KA10. Software Engineering Process	The software engineering process is central to the creation of reliable and cost-effective software products. Software engineers face a challenging and evolving highly technological landscape, fast-moving technology (sometimes to obsolescence), societal events and changes, uncertainty, a growing complexity of systems, and the trend towards digitalization of many and different domains. This is a point of no return that has facilitated the ongoing adoption of Agile and DevOps. The software engineering process will continue to evolve, profiting from advances in the engineering dimension, new tools, and the latest knowledge from the other KAs.	Students need to be acquainted with new fundamentals, new approaches, and new tools so that they can cope with this landscape. Now more than ever, software engineering process definition and management means combining rigor and flexibility, and the introduction of the broad use empirical measures to support decision making and process monitoring. The relevance of the interactions between this KA and other KAs, therefore, increases.	Critical thinking, judgment, and decision-making about different process models and tools for large, complex, and uncertain specific scenarios will be needed. Software engineers need to be skilled in defining and in assessing and improving software life-cycle processes. More than ever, software engineers must be able to learn about the process in execution and the product. And all this knowledge must be deployed in subsequent processes, and be part of the (management) decision making, possibly to avoid or overcome obstacles. This will influence professional training.
KA11. Software Engineering Models and Methods	Models have been updated and accurately classified by type. Relatively new methods, including aspect-oriented development, as well as fundamental model-driven and model-based methods, have also been introduced. Agile methods have been extended a great deal to incorporate modern techniques, such as lean development, as well as large-scale and enterprise agile methods. On this note, DevOps and release engineering have also been introduced to clarify the release aspect of agile methods.	Fundamental modeling principles and basic model types are essential and should be addressed since these are on the basis of almost any software engineering activity, regardless of the types of software engineering methods to be taught, but the key to succeed while teaching models and methods is the practice, it is essential to use project-based technique so students can feel models and methods alive. Furthermore, method types, as well as details of a few methods (particularly an agile method with DevOps in an era of agile), may be selectively addressed.	In addition to fundamental modeling principles and model and method types, some methods, particularly agile and prototyping methods with DevOps, should be addressed in a fast-moving and agile era. Furthermore, practitioners should understand the basics of model analysis, such as analysis for consistency and traceability, to ensure that a team of professionals can successfully develop, operate and maintain large and/or complex software systems.

TABLE III
SWEBOK EVOLUTION AND IMPACTS ON HIGHER EDUCATION AND TRAINING: KA12–KA18

KA	Point of evolution	Impact on higher education	Impact on professional training
KA12. Software Quality	This chapter was overhauled for alignment with notions of processes/product quality. It now includes new topics: (1) Software Dependability and Integrity Levels, (2) Standards, Models and Certifications, (3) Policies, Processes and Procedures, and (4) Quality Control and Testing.	Students must be made aware that software quality extends well beyond software testing alone. Insofar as it can help avoid defects in the first place and thus vastly reduce the need for testing resources, process quality is a key insight. Analysis of real-world case studies where broader approaches to quality have been successfully applied can be critical.	The learner is expected to have some real-world, on-project experience already. Retrospective analysis of how past projects might have been improved through better application of quality techniques can be useful. It will be important to emphasize that the cost (i.e., effort) invested in quality techniques is easy to assess, whereas the return on investment is more difficult to evaluate because it comes in the form of cost (i.e., effort) avoided—work not done— (i.e., defects were avoided entirely or found earlier when they were much cheaper to repair).
KA13. Software Security	This is a new knowledge area that focuses on the broader topics of security, particularly security fundamentals, security management, security tools, and domain-specific security, as well as major security engineering activities (such as security testing) that were briefly described under the Computing Foundations KA in the last SWEBOK edition. The existence of a separate KA emphasizes that security should be a first-class quality attribute in any development since almost any software system is connected to others, resulting in increased security risks.	Students should study this KA to learn about security fundamentals since the development of almost any software system needs to consider security concerns in the era of IoT and connected software. Furthermore, students should learn basic security engineering techniques aligned with the development life cycle, including security requirements, design, construction, and testing, to understand how security measures should be incorporated into the major development activities.	Apart from security fundamentals and major engineering activities, practitioners should be aware of available practical security solutions, including security tools and patterns. Security patterns provide guidelines to improve security characteristics such as confidentiality, integrity, and availability since security patterns incorporate the knowledge of security experts. Domain-specific security techniques should also be recognized since security is a particularly tough objective for cloud, IoT, and machine learning applications. Practitioners involved in organization, product, and project management should refer to the security management and organization section to learn the importance of systematically weaving security governance and management into their cultural and organizational behaviors.
KA14. Software Engineering Professional Practice	Additions and revisions address the following areas: (1) Professional practices following generally accepted practices, standards, and guidelines set forth by the applicable professional societies, (2) UI/UX inclusive design, (3) Considerations on diversity and inclusion, and (4) Considerations on agility.	Students should engage in certification or qualification programs offered by professional societies or national organizations. Students are expected to participate in more programs organized by professional societies to get acquainted with state-of-the-art practices apart from academic accomplishments.	Professional practices are progressing at an ever faster pace as software use becomes more widespread and is adopted in socially critical applications. Participation in the various activities of professional societies provides continuous reskilling. Software used in more critical applications requires the enforcement of a code of ethics and code of conduct, which calls for continuous training with respect to the guidelines of professional societies.
KA15. Software Engineering Economics	Focuses on the essence of engineering economics, the art of making decisions. Broaden the more traditional, purely financial view of engineering economics. Value does not always derive from money alone; that is, value can also derive from "unquantifiables" like corporate citizenship, employee well-being, environmental friendliness, customer loyalty, and so on. And more systematically address pre-project decisions, where a project is not under development but is being envisioned.	The software engineering economics (SEE) vision must be broader in two ways. It should (1) focus on the essence of engineering economics, which means the art of making decisions, and (2) extend decision-making to the pre-project phase, where the project is being envisioned, not developed. Of the software engineering teaching methods identified by [5], collaborative learning is one of the methods that could be very useful for understanding SEE, because sound decision-making depends on listening to and understanding stakeholders. This is followed by collaborative prioritization in order to make the best decision using existing methods that can be taught in a flipped classroom, where the teacher can discuss with the students the decision-making methods reported in the previous readings sent to the students about the methods to be discussed.	Decision-making based on cyclical prediction instead of linear patterns can be an effective way to apply these techniques retrospectively to the practitioner's real-world project experiences. For example, a learner could calculate the actual PW(i) of a completed project or analyze whether the specific projects implemented in a digital transformation really were the best.
KA16. Computing, KA17. Mathematical, KA18. Engineering Foundations	Minor improvements in topic presentation. Discussions of Artificial Intelligence and Machine Learning were added. Furthermore, the revision includes a deeper coverage of measurement, particularly its implications for programming languages, and a more comprehensive discussion of root cause analysis.	Collaborative learning, where students teach each other and evaluate other students' understanding, can improve comprehension and also enhance cooperation and teamworking skills.	The challenge is that many practitioners may not recognize the need for theoretical foundations (e.g., "I have been successful in my career so far without that"). It is critical to tie foundational knowledge in with how it affects real-world situations.

TABLE IV
SPECIFIC CUTTING-EDGE CONCEPTS IN SWEBOK V4

Concern	Related KA	Impact on higher education	Impact on professional training
Agile & DevOps	KA1, KA9, KA10, KA11	Agile and DevOps have a direct impact on several, and an indirect impact on all, KAs. Agile and DevOps fit the current scenario: fast-moving and changing times, full of uncertainty. Agile and DevOps must be explained, highlighting their fundamentals, as they constitute paradigm shifts rather than just a new set of practices. It will help to present these new methods together with other approaches to develop critical thinking and decision-making, and resource optimization/ management in the case of DevOps.	Agile and DevOps training should open up new perspectives for professionals. One is critical thinking and judgment based on DevOps values/principles/practices and schemes. DevOps affects decision-making at all levels of the software engineering process and KAs. Agile impacts both the engineering and the management level.
AI and SE	KA16	The recent resurge in artificial intelligence (AI), and its prospects have been swift and strong. In the case of AI systems, knowledge has very often not yet been consolidated. For this reason, AI is only considered within the Computing Foundations KA in SWEBOK V4. More and more systems will be using AI in the coming years, and humans will depend on these systems in many ways. Higher education programs can introduce students to emerging endeavors on AI for SE and SE for AI as a part of the future of software engineering by referring to the Computing Foundations KA. Future SWEBOK versions will address how to engineer systems that include AI subsystems. AI is likely to change how software is engineered, and this will also be addressed by future versions of SWEBOK.	Professional training programs should address emerging automated techniques and tools by referring to the general introduction of AI and ML, as well as AI for SE. Practitioners should understand how such AI techniques and tools are going to replicate particular developer behavior, ranging from resolving ambiguous requirements to predicting maintainability. Furthermore, professionals should understand the need for particular software engineering support for AI since AI systems are developed differently from traditional software systems, as the rules and system behavior of AI systems are inferred from training data rather than written down as program code. Professionals can learn from emerging recommended software engineering practices for AI, which are often formalized as patterns.

TABLE V
LINKING TEACHING METHODS IN SE AND SWEBOK V4 KAS

Teaching method	Knowledge Area
Project-Based Learning	KA1, KA3, KA4, KA6, KA7, KA8, KA9, KA10, KA12
Learning by Reflection	KA2, KA3, KA9, KA10, KA11, KA16, KA17, KA18
Just-in-Time Learning	KA2, KA14
Participation in SW Community	KA4, KA5
Flipped Learning	KA15, KA10
Experimental/Research-Based Learning	KA5
Global Software Development	KA4
Problem-Based Learning	KA4, KA5, KA11, KA13
Active Learning/Learning by Doing	KA5, KA8, KA10
Collaborative/Peer Learning	KA15
Agile Learning	KA9

as possible in developing distributed, networked systems for use by the public at large, and the need for specialized skills to be able to work effectively in these areas.

III. TRAINING AND EDUCATION IN DIGITAL ERA

Tables I, II, and III also show the main impact of each KA on both higher education and professional training. Moreover, Table V shows the teaching methods, identified in recent work by [5], that best suit each KA. Software engineering education and training programs are expected to focus on core topics for each SWEBOK V4 knowledge area as described in the Tables I–III, through suggested teaching methods identified in Table V which maps KAs to teaching methods.

Table IV includes some cutting-edge concepts and how they are addressed in SWEBOK V4. “Agile & DevOps” are two topics that should be incorporated into any education and training program as they are practical fundamentals that are well accepted by industry and should be addressed by academia. However, in and of themselves, it is not felt that they should constitute a key area but are applicable across multiple of the existing key areas in SWEBOK V4. The “AI and SE” connection, while not a key area of SWEBOK V4 either, is a topic that should be considered and developed as an advanced topic in the various software engineering programs.

IV. CONCLUSION

In this paper, we described the evolution of SWEBOK and its impact on higher education and training. Although all the teaching methods identified by [5] can be applied to most KAs, Table V summarizes the ones that are most suitable for each KA. The software engineering profession needs to be aware of several specific methods to instruct the different KAs. The emergence of Architecture, Operations and Security KAs requires specific reflection on their connections with SE2014 [6] already covered by SWEBOK V3 [4]. A special attention is also required to update SWECOM [7] to incorporate potential competencies required to perform new KAs.

ACKNOWLEDGMENT

The authors thank the SWEBOK editors, IEEE CS staff, and other volunteers for their contributions to the SWEBOK development. We thank Rich Hilliard, Alain April, Xin Peng, Katsutoshi Shintani, Eda Marchetti and Said Daoudagh for their support and contributions to this paper.

REFERENCES

[1] “SWEBOK guide v4 beta,” https://waseda.app.box.com/v/ieee-cs-swebok, (Accessed on 22/03/2023).
[2] P. Bourque and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK): Version 3.0.* IEEE Computer Society, 2014.
[3] P. Kamthan and H. Washizaki, “SWEBOK matters: Report and reflection of a seke panel on the educational and professional implications of SWEBOK,” *Int. J. Softw. Eng. Knowl. Eng.*, vol. 32, no. 11–12, 2022.
[4] A. Alarifi, M. Zarour, N. Alomar, Z. Alshaikh, and M. Alsaleh, “SECDEP: software engineering curricula development and evaluation process using SWEBOK,” *Inf. Softw. Technol.*, vol. 74, pp. 114–126, 2016.
[5] K. P. Anicic and Z. Stapic, “Teaching methods in software engineering: A systematic review,” *IEEE Softw.*, vol. 39, no. 6, pp. 73–79, 2022.
[6] “Curriculum guidelines for undergraduate degree programs in software engineering (SE2014),” https://www.acm.org/binaries/content/assets/education/se2014.pdf, 2015, (Accessed on 26/05/2023).
[7] “Software engineering competency model (SWECOM),” http://www.dahlan.web.id/files/ebooks/SWECOM.pdf, 2014.