A Combined Knowledge and Competency (CKC) Model for Computer Science Curricula

By Amruth N. Kumar, *Ramapo College of New Jersey*, Brett A. Becker, *University College Dublin*, Marcelo Pias, *Federal University of Rio Grande (FURG)*, Michael Oudshoorn, *High Point University*, Pankaj Jalote, *Indraprastha Institute of Information Technology*, Christian Servin, *El Paso Community College*, Sherif G. Aly, *American University in Cairo*, Richard L. Blumenthal, *Regis University*, Susan L. Epstein, *Hunter College and The Graduate Center of The City University of New York*, and Monica D. Anderson, *University of Alabama*

All prior curricular guidelines for computer science have used a knowledge model, which consists of knowledge areas, knowledge units within the knowledge areas, and learning outcomes for the topics within those knowledge units. More recently, competency models have been explored for curricular guidelines. A competency model consists of competency specifications that list the knowledge, skills and dispositions needed to complete tasks. Both knowledge models and competency models have their benefits and shortcomings. We propose a model for computer science curricular guidelines that synergistically combines knowledge and competency models, in particular, the knowledge model last proposed in CS2013 [1] and the CoLeaf competency model last proposed in an ITiCSE working group report [8,11], both modified to facilitate integration. The combined model called CKC emphasizes both ends of the learning continuum and facilitates teaching as well as evaluation. It provides both an epistemological and teleological perspective of computer science content. We provide instructions for designing computer science curricula using the CKC model.

INTRODUCTION

Over the last decade, the focus of curricular design has been changing from what is taught to what is learned. What is taught is traditionally referred to as a knowledge model of the curriculum and what is learned is referred to as a competency model of the curriculum. One of the early efforts to design a competency model of a curriculum was for Information Technology with IT2017 guidelines [23]. This was followed by an ITICSE Working group effort to model competencies for computing education in general [11], and the Computing Curricula CC2020 report [6] which proposed a competency model for various computing disciplines, Computer Science, Information Systems, and Data Science among them. On the heels of CC2020, competency models of curricula for Information Systems 2020 [19] and Data Science 2021 [9] were developed. CS2013, the most recent curricular guidelines for computer science [1] utilized a knowledge model of the curriculum. Since then, an ITiCSE working group has tried to design sample competency statements for computer science [8]. A process was also proposed for converting a knowledge model to a competency model for computer science [7].

Computer science encompasses all aspects of solving problems with computers. Given the expansive nature of the discipline, applying a competency model to it is challenging. Yet, given the current demands for increased accountability in higher education, both by society at large and by accrediting bodies (e.g., [21]) it behooves us to seriously contemplate a competency model for computer science, despite the challenges. It is in this context that we explore the relative benefits and shortcomings of knowledge and competency models, and propose the CKC model that combines the two for computer science curricula.

WHAT IS COMPETENCY?

Competency was defined as the sum of knowledge, skills, and dispositions in IT2017 [23] wherein, dispositions are defined as cultivable behaviors desirable in the workplace [21]:

Competency = Knowledge + Skills + Dispositions

In CC 2020 [6], competency was further elaborated as the sum of the three within the performance of a task. Instead of the additive model of IT 2017, CC2020 defined competency as an intersection of the three:

Competency = Knowledge ∩ Skills ∩ Dispositions

More recently, a projective model of competency was proposed wherein competence is a point in a 3D space with knowledge, skills, and dispositions as the three axes of the space (Figure 1) [21]. In this model, all three are required for proper execution of a task. One does not "build up" to competence by adding dispositions to knowledge and skills. Instead, when one talks about knowledge or skills or dispositions individually, one projects the competency point in the 3D space to one of the axes, temporarily ignoring the other two. Speaking of only knowledge, skills or dispositions is not denying the importance of the other two, but de-emphasizing them for temporary effect.

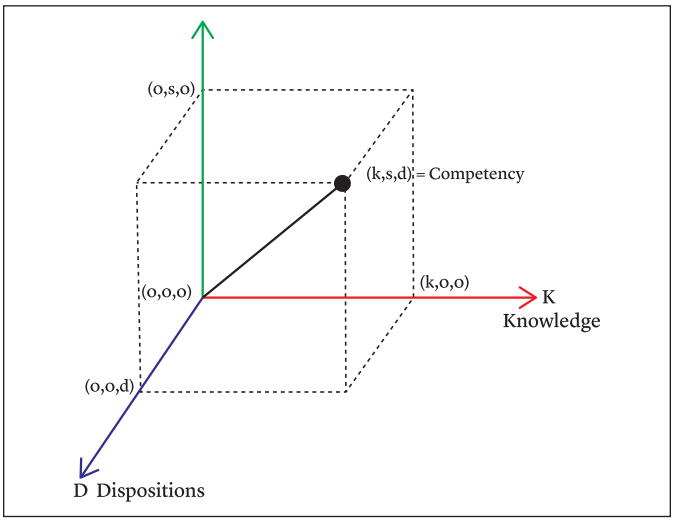

ILLUSTRATION: ©2023 AMRUTH N. KUMAR

Figure 1: Projective model of competency [21].

Since the primary difference between knowledge models and competency models is the latter's explicit emphasis on dispositions, an understanding of how dispositions can be fostered is necessary to realize the added benefits of using competency models.

KNOWLEDGE MODEL VERSUS COMPETENCY MODEL

Knowledge models of curricula that started as a listing of knowledge areas and knowledge units, i.e., what should be learned, were gradually extended in computer science to also include skills, i.e., how to apply the learning. The most recent knowledge model of computer science curricula, viz., CS2013 [1] also mentions the importance of dispositions, albeit in passing.

Competency models consist of competency specifications, with each specification consisting of a competency statement and the knowledge, skills and dispositions needed to complete the task stated in the competency statement. An advantage of a competency model is its explicit emphasis on dispositions.

In computing education, research has been conducted on associating dispositions (sometimes referred to as personality traits in literature) with performance metrics in software engineering [27]. Studies have been conducted, both among working professionals [3,18,22,26,27] and students [14,24,29] to investigate issues such as predicting performance in pair programming, forming optimal teams, and finding the best fit for specific work roles. Nonetheless, fostering dispositions is not as well understood and remains a target of ongoing research.

While research has been conducted on soft skills such as communication (written, oral), teamwork and management skills in computing education (e.g., [4,13,20]), dispositions are different from soft skills, even when similarly worded (e.g., collaborative) in that dispositions involve the willingness and intent to apply skills in a given context [10,20,25]. Moreover, dispositions are habitual, not one-off behaviors [20,25].

Since the primary difference between knowledge models and competency models is the latter's explicit emphasis on dispositions, an understanding of how dispositions can be fostered is necessary to realize the added benefits of using competency models. The current understanding of dispositions is that they are observable and learnable, but not necessarily teachable. They can be formatively modeled for students, but not necessarily summatively assessed, i.e., they can be qualitatively promoted using activities such as reflection (e.g., [15,16]), but not necessarily quantitatively measured [21].

Another difference between knowledge models and competency models is their initial focus.

- Knowledge models start with topics organized as knowledge areas and knowledge units and end with expected learning outcomes that are measurable.
- Competency models start with competencies that are observable in the accomplishment of tasks, and end with identifying the topics and knowledge units needed to accomplish them.

This difference is depicted in Figure 2. Note that whereas learning outcomes in the knowledge model are measurable, competencies in the competency model are only observable [8].

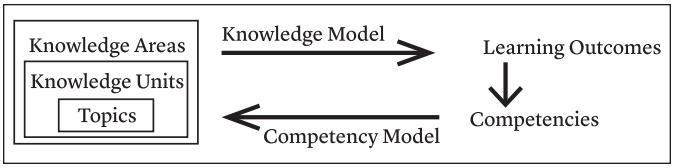

Figure 2: Knowledge model versus Competency model.

Using Artificial Intelligence (AI) parlance, knowledge model is forward reasoning (data → goal) whereas competency model is backward reasoning (goal → data). In AI, neither approach is considered superior to the other. Whether to use forward reasoning (e.g., when solving a crossword puzzle) or backward reasoning (e.g., when solving a maze) is determined by branching factor—the number of options that must be considered at each step in either direction. CS 2013 identified 18 knowledge areas in computer science containing a total of 163 knowledge units. The number of tasks that a computer science graduate may be called upon to complete in the workplace (not the number of possible jobs the graduate might be able to fill) on the other hand can be in the hundreds or even thousands, far exceeding the number of knowledge areas or knowledge units. Then again, the number of tasks to which a knowledge area can contribute is far greater than the number of knowledge areas that contribute to a task. So, knowledge model has fewer starting points (knowledge areas/knowledge units), but larger branching factor (tasks to which a knowledge area can contribute), whereas competency model has far more starting points (number of tasks), but smaller branching factor (number of knowledge areas/units that contribute to a task).

KNOWLEDGE MODEL OR COMPETENCY MODEL FOR COMPUTER SCIENCE?

A knowledge model organizes content into knowledge areas, which are silos of related content. Each knowledge area consists of multiple knowledge units, and each knowledge unit consists of multiple topics. This epistemological organization of content facilitates the process of designing courses and curricula: multiple courses may be carved out of a single knowledge area and a course may draw content from multiple knowledge areas. Therefore, a knowledge model with its initial emphasis on knowledge areas well serves the needs of teaching. The same cannot be said about a competency model which distributes

A competency model attempts to fix this by placing initial emphasis on outcomes, and identifying the curricular topics and dispositions needed to achieve the outcomes. It is the curricular equivalent of test-driven development: understanding what one needs to be able to do before learning how to do it. This approach can be highly motivating, especially for goal-driven students.

content across competency specifications, making it harder for a novice educator to see the forest for the trees. In addition to being repetitive (the same concept listed in multiple competency specifications), competency-oriented organization of content loses important relationships among topics such as generalization, aggregation, classification and grouping, which are essential for a deeper understanding of the discipline.

Nonetheless, knowledge models of computer science curriculum do not seem to have lived up to expectations when it comes to evaluation of learning. To wit, it is now standard practice for industry to use coding interviews to recruit computer science graduates. This highlights a drawback of knowledge models of the curriculum: educators who consult them often end up emphasizing content over outcomes.

A competency model attempts to fix this by placing initial emphasis on outcomes, and identifying the curricular topics and dispositions needed to achieve the outcomes. It is the curricular equivalent of test-driven development: understanding what one needs to be able to do before learning how to do it. This approach can be highly motivating, especially for goal-driven students.

An argument against a competency model is that it is intractable. A competency specification encompasses a subset of topics. CS2013 lists 163 knowledge units, each in turn containing multiple topics. Even if we were to consider competency specifications as encompassing subsets of knowledge units instead of topics, theoretically, we can list 2163 competency specifications for computer science! The same could be said for other disciplines also. Pairing topics with skill levels as recommended [8] would enlarge the space of possible competency specifications even more. So, a competency model can never be comprehensive even for a given knowledge model.

Another argument against a competency model is that emphasizing outcomes ahead of content reduces a discipline to job-training, e.g., computer science is reduced to a vocational

discipline that prepares students for a laundry list of well-defined tasks. This is unfortunate given the increasing recognition of computational thinking as one of the fundamental skills of the 21st century [28], and the central role played by computer science in inculcating it among students regardless of their major.

So, the choice between a knowledge model and a competency model for computer science may be seen as a choice between viewing computer science as a scientific discipline that promotes problem-solving and computational thinking versus a technical discipline that trains students to solve problems for the workplace; one that helps students learn to think in the long term versus one that prepares them to act in the near term.

It bears stating that this dichotomy is not a feature but rather a bug of knowledge model versus competency model – sole emphasis on either content or outcomes is not inherent to either model, but rather, it is what a typical educator takes away from them. After all, knowledge models do include learning outcomes and competency models do list content in each competency specification.

A knowledge model with its initial emphasis on content and a competency model with its initial emphasis on outcomes are complementary views of the same learning continuum, as depicted in Figure 2. We propose a curricular model called CKC that synergistically combines the two and offers the benefits of both.

- By canonically listing concepts, organizing them into knowledge areas and knowledge units and making explicit, relationships among them such as generalization, aggregation, classification and grouping, a knowledge model facilitates an educator's job of organizing related concepts into coherent courses and curricula.
- By grouping content needed for each competency specification, a competency model helps a learner make associations among complementary concepts from multiple knowledge areas. By explicitly listing the tasks a graduate should be expected to complete, it also facilitates evaluation of student learning and of programs.

So, a knowledge model facilitates teaching whereas a competency model facilitates evaluation. By placing emphasis on both ends of the learning continuum, the combined model can help educators with both teaching and evaluation. It can help learners gain both epistemological (how topics are related to each other) and teleological (the utility of each topic) perspectives of content.

We conclude that:

- neither model is a substitute for the other;
- both the models have their advantages and shortcomings; and
- knowledge models and competency models complement each other, and work better considered together than apart.

THE DESIGN OF CKC MODEL FOR COMPUTER SCIENCE CURRICULA

We propose CKC as model of computer science curricula that combines the knowledge model specified in CS2013 [1] and the CoLeaf competency model proposed for computer science in recent literature [7,8,11,12]. Recall that the CS2013 knowledge model is specified in terms of 18 knowledge areas, each broken down into knowledge units consisting of multiple topics. Learning outcomes are specified for each knowledge unit. The CoLeaf competency model for computer science [8,11] proposes a hierarchy of competency specifications. Each competency specification contains a vernacular description called the competency statement and enumeration of a subset each of topics, skills and dispositions needed to complete the task described in the competency statement. In order to facilitate a synergistic integration of the two models, we start by proposing changes to both the models.

CORE TOPICS IN KNOWLEDGE MODEL

In CS2013, core hours were defined along two tiers: Tier I (165 hours) and Tier II (143 hours). Computer science programs were expected to cover 100% of Tier I core topics and at least 80% of Tier II topics. While proposing this scheme, CS2013 was mindful that the number of core hours has been steadily increasing in curricular recommendations, from 280 hours in CC2001 [17] to 290 hours in CS2008 [5] and 308 hours in CS2013 [1]. Not all computer science programs may be able to accommodate the increasing number of core topics in their curricula.

We propose a sunflower model of core topics wherein topics are designated as:

- Computer Science (CS) core—topics that every computer science graduate must know; and
- Knowledge Area (KA) core—topics that any coverage of a knowledge area must include.

This model acknowledges that often, the design of curricula in computer science programs is constrained by regional needs, credit limitations, local availability of instructional expertise, and/or historical evolution of programs. While all the programs must cover CS core topics, a program may choose to cover some knowledge areas in greater depth/breadth than other knowledge areas. In Figure 3, highlighting shows such selective coverage of knowledge areas in a typical computer science program.

When coherently chosen, the knowledge areas covered by a computer science program will constitute the program's competency area. Some possible competency areas are:

- **Software**, consisting of the knowledge areas: Software Development Fundamentals, Algorithmic Foundations, Foundations of Programming Languages and Software Engineering.
- **Systems**, consisting of some of the following knowledge areas: Systems Fundamentals, Architecture and Organization, Operating Systems, Parallel and Distributed Computing, Networking and Communication, Security and Data Management.
- **Applications**, consisting of some of the following knowledge areas: Graphics and Interactive Techniques, Artificial Intelligence, Specialized Platform Development, Human-Computer Interaction, Security and Data Management.

Note that the software competency area is a pre-requisite of the other two competency areas, which is in keeping with the hierarchical structure proposed in the CoLeaf model [11].

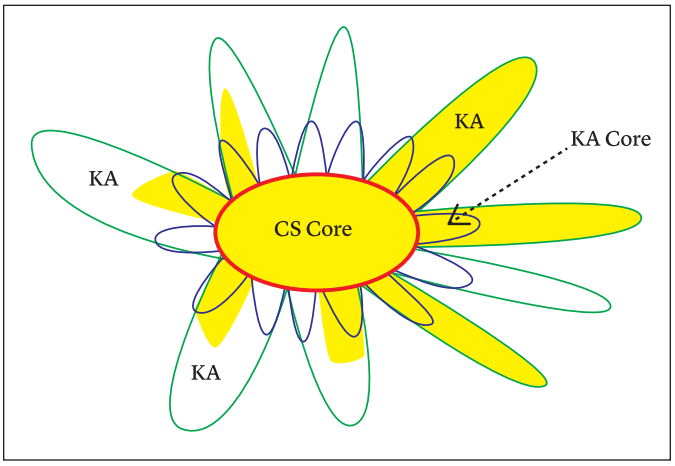

Figure 3: Sunflower model of core topics.

CHANGES TO THE COLEAF COMPETENCY MODEL

We propose the following changes to the CoLeaf competency model [8,11].

- We extract the task from the competency statement and state it separately in a competency specification. A task is what an employer might want done whereas a competency statement specifies what a graduate might bring to bear in terms of knowledge, skills and dispositions to complete the task. A task is objective whereas the ways in which it can be accomplished can be subjective. Separating out the task will make it easier to adapt a competency statement to local conditions by accommodating how the task is locally accomplished.
- We focus on tasks at higher levels in Bloom's taxonomy [2]: application, analysis, evaluation and synthesis, tasks that are authentic to a workplace setting.
- Instead of repetitively listing all the topics in detail in each competency specification, we parsimoniously list the knowledge areas and knowledge units already specified in the knowledge model, thereby linking the two models together.
- Just as topics are organized in terms of knowledge areas and knowledge units in a knowledge model, we propose to organize competency specifications into competency areas and competency units - the nomenclature was intentionally chosen to mirror that of the knowledge model for the sake of consistency. Recall that competency areas (e.g., Software, Systems and Applications) were introduced earlier as the focus of computer science programs that coherently choose knowledge area coverage in their curriculum. Since the programs will already be focused on one or some of these competency areas, organizing competency specifications in terms of the very same competency areas will facilitate evaluation of the programs.
- For competency units, we chose orthogonal issues that apply to every knowledge area, such as: Design, Development, Evaluation, Maintenance, Social Acceptance, Improvement and Theory. A competency area is the sum of its competency units. Whereas the number of competency areas targeted by a program indicates its breadth, the number competency units targeted by the program in each competency area indicates its depth.

Figure 4 illustrates a sample competency specification from Software / Application competency area. Note that it includes a task, competency statement, competency areas and units to which it applies, and knowledge areas, knowledge units and skills it requires. Note that the competency specification draws upon two knowledge areas from the CS2013 knowledge model [1]: Software Development Fundamentals and Software Engineering.

- **Task:** Identify appropriate tools to assist in development, design, or debugging
- **Competency statement:** Apply knowledge of common classes of software tools (static analysis, dynamic analysis, version control, coverage, refactoring, etc.) and be able to identify problems where application of such tools would be appropriate.
- **Competency area:** Software / Application
- **Competency unit:** Development / Integration
- **Required knowledge areas and knowledge units:**
  - Software Development Fundamentals / Development Methods
  - Software Engineering / Tools and Environments
  - Software Engineering / Software Construction
  - Software Engineering / Software Verification and Validation
- **Required skill level:** Explain

Figure 4: Sample Competency Specification in Software / Applications competency area.

SPECIFYING DISPOSITIONS

The CoLeaf competency model stipulates that dispositions are an integral part of every competency specification and must be explicitly included in the competency statements [12]. While not disputing this stipulation, we consider the following points.

- Dispositions are generic to knowledge areas. Some dispositions are more important at certain stages in a student's development than others, e.g., *persistent* is important in introductory courses (Software Development Fundamentals knowledge area), whereas *self-directed* is important in advanced courses (e.g., Foundations of Programming Languages and Artificial Intelligence knowledge areas). *Collaborative* applies to courses with group projects (e.g., Software Engineering knowledge area) whereas *meticulous* applies to mathematical foundations. So, associating dispositions with knowledge areas makes it easier for the instructor to consistently promote dispositions during the accomplishment of tasks to which the knowledge area contributes, while bearing the "big picture" in mind.

ILLUSTRATION: ©2023 AMRUTH N. KUMAR

- Dispositions are not desirable behaviors exhibited one-off, but rather, *habits* displayed consistently and without coercion [20,25]. This calls for repeated exposure of students to each disposition. Associating dispositions with knowledge areas instead of the numerous competency specifications associated with each knowledge area makes the need for repeated exposure clear while keeping the model succinct.
- While there is universal consensus on the importance of dispositions for the professional success of computer science graduates, the processes and practices for fostering dispositions are not yet well understood. After all, dispositions are learnable, but may not necessarily be teachable. In the absence of clear guidelines for fostering dispositions, a light touch (stating without hammering home) may earn better buy-in from computer science educators.

Therefore, we propose to associate dispositions with knowledge areas instead of competency specifications.

DESIGN OF THE COMBINED KNOWLEDGE AND COMPETENCY (CKC) MODEL

Figure 5 illustrates CKC, a synergistic integration of the CS2013 knowledge model and the CoLeaf competency model of computer science curricula. In the figure, the knowledge model is to the left, and consists of knowledge areas such as the 18 listed in CS2013 [1]. Each knowledge area consists of 6–20 knowledge units which are themselves aggregates of topics. Also associated with each knowledge area are the dispositions most appropriate for it.

The topics in each knowledge area are classified as either CS core, KA core or non-core. CS core topics define what it means to be a computer science graduate. KA core topics determine the competency area(s) of a computer science program: Software, Systems or Applications. Competency areas are one point of interaction between the knowledge model and competency model of a curriculum.

In Figure 5, the competency model is to the right. The competency model consists of the competency areas targeted by a computer science program through the choice of knowledge areas in which it offers significant coursework. Each competency area is broken down into orthogonal competency units such as Design, Development, and Evaluation. In each competency unit, we identify a set of tasks that an employer might expect a computer science graduate to complete (e.g., "Design the architecture of a web-based service"). For each task, we identify the knowledge areas and knowledge units needed to complete the task. The knowledge units may belong to CS core, KA core or non-core. The knowledge areas and knowledge units identified for tasks are where the competency model connects back to the knowledge model. Finally, the dispositions that facilitate completion of a task are obtained from the knowledge areas identified for the task.

The relationship between tasks and competency units could be many-to-many: many competency units may map to a single task and many tasks may be identified for a competency unit. Similarly, the relationship between tasks and knowledge areas/units could be many-to-many. The tasks ideally target higher levels in Bloom's taxonomy [2].

Skill levels are the final element that bind knowledge and competency models together: they are part of both. In the CS2013 knowledge model, skill levels were associated with

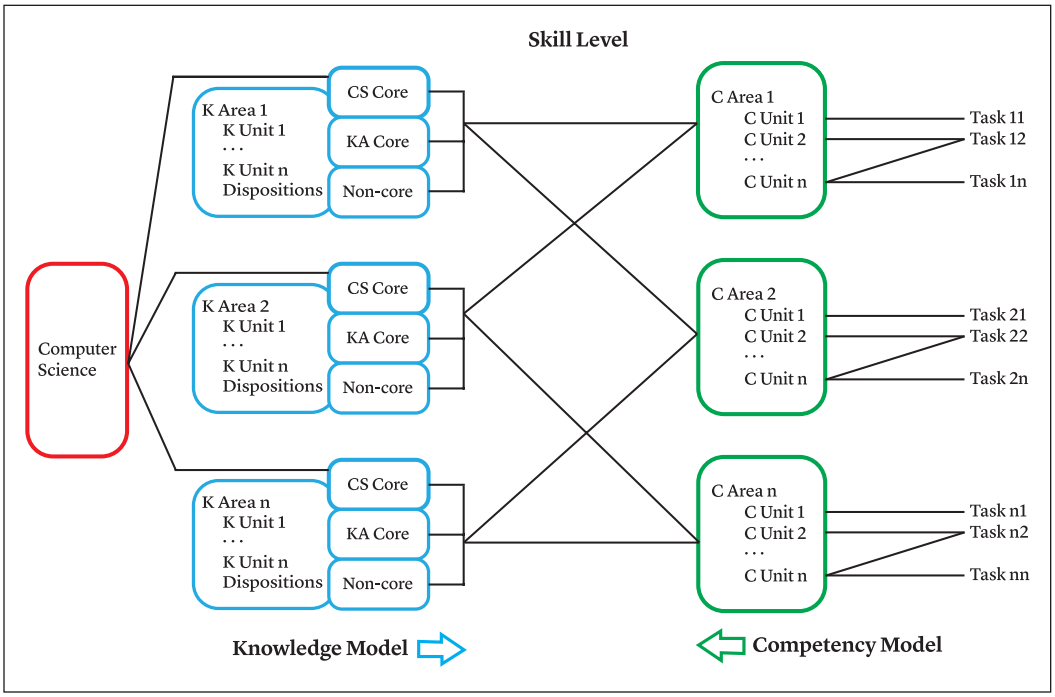

Figure 5: Combined Knowledge and Competency (CKC) Model of Computer Science Curricula.

ILLUSTRATION: ©2023 AMRUTH N. KUMAR

learning outcomes. In the CoLeaf competency model, skill levels are included in every competency specification. These skill levels are typically derived from Bloom's taxonomy [2]. We align the skill level in the knowledge model with the skill level needed to complete tasks in the competency model, as shown in Figure 5. Note that in the figure, the links lack directionality to signify that interdependencies work in either direction based on whether one starts with the knowledge model or the competency model.

To summarize the integration, competency areas are referred to in the knowledge model, knowledge areas are referred to in the competency model, skill levels provide alignment between the two models and dispositions, the raison d'etre of the competency model are associated with knowledge areas in the knowledge model, but used to facilitate completion of tasks specified in the competency model.

USING THE COMBINED MODEL TO DESIGN A CURRICULUM

We proposed CKC as a combined knowledge and competency model for computer science curricula that caters to both ends of the learning continuum: teaching and evaluation. We propose the following procedure for creating the curriculum of a computer science program from the CKC model.

1. Design the courses and curricula using the knowledge areas and knowledge units of the CKC model.
2. Based on the knowledge areas chosen to be covered in the curriculum, identify the competency area(s) targeted by the curriculum.
3. Select or adapt the tasks listed in the CKC model for the competency units in those competency areas.
4. Create or modify the competency statements for those tasks in consultation with local stakeholders (academics, industry representatives, policy makers, etc.) [11]; and use the competency statements to evaluate outcomes of the program.
5. In a cycle of continual improvement, repeat steps 1–4 to improve courses, competency statements, and outcomes of the program.

Step 1 in the process promotes standardization of a program and facilitates comparison of programs. Step 2 promotes individualization of the program—what sets it apart from other programs. Steps 3 and 4 customize the program to meet local needs, an essential part of any program design. The cycle of continual improvement in step 5 helps maintain the currency and vitality of a program.

Having designed the CKC model, we are currently implementing it for computer science curricula. To that end, we have identified dispositions applicable to each knowledge area identified in CS2013 [1] and drafted competency specifications for all the knowledge areas in the format shown in Figure 4. Future work includes identifying competency specifications that transcend individual knowledge areas in each competency area and short-listing competency specifications that rely solely on CS and KA core topics. ❖

We proposed CKC as a combined knowledge and competency model for computer science curricula that caters to both ends of the learning continuum: teaching and evaluation. We propose the following procedure for creating the curriculum of a computer science program from the CKC model.

Acknowledgements

The authors gratefully acknowledge the contributions of the other members of the CS2023 task force: Eric Eaton, Michael Goldweber, Douglas Lea, Rajendra K. Raj, Susan Reiser, Titus Winters and Qiao Xiang. Partial support for this work was provided by the National Science Foundation under grant DUE-2231333.

References

1. ACM/IEEE-CS Joint Task Force on Computing Curricula. Computer Science Curricula 2013. Technical Report. (ACM Press and IEEE Computer Society Press, 2013). https://doi.org/10.1145/2534860
2. Bloom, B.S., Engelhart, M. D., Furst, E. J., Hill, W. H. and Krathwohl, D. R. *Taxonomy of educational objectives: The classification of educational goals. Vol. Handbook I: Cognitive domain*. New York: David McKay Company (1956).
3. Calefato, F., Lanubile, F. and Vasilescu, B. A large-scale, in-depth analysis of developers' personalities in the Apache ecosystem. *Inf. Softw. Technol.*, 114, (2019), 1–20.
4. Carr, M. and Claxton, G. Tracking the Development of Learning Dispositions. *Assessment in Education: Principles, Policy & Practice* 9,1, (2002), 9-37.
5. Cassel, L., Clements, A., Davies, G., Guzdial, M., McCauley, R., McGettrick, A., Sloan, R., Snyder, L., Tymann, P. and Weide. B.W. Computer Science Curriculum 2008: An Interim Revision of CS 2001. Technical Report. (ACM Press, New York, NY, USA, 2008).
6. Clear, A., Parrish, A., Impagliazzo, J., Wang, P., Ciancarini, P., Cuadros-Vargas, E., Frezza, S., Gal-Ezer, J., Pears, A., Takada, S., Topi, H., van der Veer, G., Vichare, A., Waguespack, L. and Zhang, M. Computing Curricula 2020 (CC2020): Paradigms for Future Computing Curricula. Technical Report. (Association for Computing Machinery / IEEE Computer Society, New York, NY, USA, 2020).
7. Clear, A., Clear, T., Impagliazzo, J. and Wang, P. From Knowledge-based to Competency-based Computing Education: Future Directions. In 2020 IEEE Frontiers in Education Conference (FIE). (IEEE, New York, 2020), 1–7.
8. Clear, A., Clear, T., Vichare, A., Charles, T., Frezza, S., Gutica, M., Lunt, B., Maiorana, F., Pears, A., Pitt, F., Riedesel, C. and Szynkiewicz, J. Designing Computer Science Competency Statements: A Process and Curriculum Model for the 21st Century. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '20)*. (Association for Computing Machinery, New York, NY, USA, 2020), 211–246.
9. Danyluk, A. and Leidig, P. Computing Competencies for Undergraduate Data Science Curricula (DS2021). Technical Report. (Association for Computing Machinery, New York, NY, USA, 2021).
10. Diez, M.E. and Raths, J. (eds.). *Dispositions in Teacher Education*. (Information Age Publishing, Inc. Charlotte, NC, 2007).
11. Frezza, S., Daniels, M., Pears, A., Cajander, A., Kann, V., Kapoor, A., McDermott, R., Peters, A., Sabin, M. and Wallace, C. Modelling Competencies for Computing Education beyond 2020: A Research Based Approach to Defining Competencies in the Computing Disciplines. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (Larnaca, Cyprus) (ITiCSE 2018 Companion)*. (Association for Computing Machinery, New York, NY, USA, 2018), 148–174.
12. Frezza, S., Clear, T. and Clear, A. Unpacking Dispositions in the CC2020 Computing Curriculum Overview Report, *2020 IEEE Frontiers in Education Conference* (FIE), Uppsala, Sweden, 2020), 1-8.
13. Groeneveld, W., Becker, B.A. and Vennekens, J. Soft Skills: What do Computing Program Syllabi Reveal About Non-Technical Expectations of Undergraduate Students? In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20)*. (Association for Computing Machinery, New York, NY, USA, 2020), 287–293.

14. Gulati, J., Bhardwaj, P., Suri, B., and Lather, A.S. A study of relationship between performance, temperament and personality of a software programmer. *Software Engineering Notes* 41,1 (Feb. 2016), 1–5.
15. Hazzan, O. and Har-Shai, G. Teaching and learning computer science soft skills using soft skills: the students' perspective. In *Proceedings of the 45th ACM technical symposium on Computer science education (SIGCSE '14)*. (Association for Computing Machinery, New York, NY, USA, 2014), 567–572.
16. Hazzan, O. and Har-Shai, G. Teaching computer science soft skills as soft concepts. In *Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13)*. (Association for Computing Machinery, New York, NY, USA, 2013), 59–64.
17. Joint Task Force on Computing Curricula. Computing Curricula 2001. *Journal of Educational Resources in Computing*. 1, 3 (Sept. 2001), 1–es. https://doi.org/10.1145/384274.384275
18. Kanij, T., Merkel, R. and Grundy, J. An empirical investigation of personality traits of software testers. In *Proceedings of the 8th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '15)*, (July 2015), 1–7.
19. Leidig, P. and Salmela, H. *A Competency Model for Undergraduate Programs in Information Systems (IS2020). Technical Report*. (Association for Computing Machinery, New York, NY, USA, 2021).
20. Perkins, D.N., Jay, E. and Tishman, A. Beyond Abilities: A Dispositional Theory of Thinking. *Merrill-Palmer Quarterly* 39,1 (1993), 1–21.
21. Raj, R.K., Kumar, A.N., Sabin, M., and Impagliazzo, J. Interpreting the ABET Computer Science Criteria Using Competencies. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2022)*. (Association for Computing Machinery, New York, NY, USA, 2022), 906–912.
22. Rastogi, A. and Nagappan, N. On the personality traits of github contributors. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE '16)*, (IEEE Computer Society, 2016), 77–86.
23. Sabin, M., Alrumaih, H., Impagliazzo, J., Lunt, B., Zhang, M., Byers, B., Newhouse, W., Paterson, W., Tang, C., van der Veer, G. and Viola, B. *Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology*. (Association for Computing Machinery, New York, NY, USA, 2017).
24. Salleh, N., Mendes, E., Grundy, J. and St. J. Burch, G. An empirical study of the effects of conscientiousness in pair programming using the five-factor personality model, in *Proceedings of the International Conference on Software Engineering*, 1, (2010), 577–586.
25. Schussler, D.L. Defining Dispositions: Wading Through Murky Waters. *The Teacher Educator* 41,4 (2006), 251–268.
26. Stewart, C., Marciniec, S., Lawrence, D. and Joyner-McGraw, L. Thinkubator approach to solving the soft skills gap. *American Journal of Management* 20,2, (2020), 78-89.
27. Varona, D. and Capretz, L.F. Assessing a candidate's natural disposition for a software development role using MBTI. In *Psychology of Programming Interest Group (PPIG) 31st Annual Workshop*, (2020), 1–7.
28. Wing, J.M. Computational thinking. *Communications of the ACM* 49,3 (2006), 33–35.
29. Xu, B., Zhang, Q., Gao, K., Yu, G., Zhang, Z. and Du, Y. Recognition of learners' personality traits for software engineering education, in *ACM International Conference Proceeding Series*, (July 2021), 1–7.

Amruth N. Kumar
Ramapo College of New Jersey
505 Ramapo Valley Road
Mahwah, NJ 07430
amruth@ramapo.edu

Brett A. Becker
University College Dublin
Dublin, Ireland
brett.becker@ucd.ie

Marcelo Pias
Federal University of Rio Grande (FURG)
Brazil
mpias@furg.br

Michael Oudshoorn
High Point University
High Point, NC, USA
moudshoo@highpoint.edu

Pankaj Jalote
Indraprastha Institute of Information Technology
Delhi, India
jalote@iiitd.ac.in

Christian Servin
El Paso Community College
El Paso, TX, USA
cservin1@epcc.edu

Sherif G. Aly
American University in Cairo
Cairo, Egypt
sgamal@aucegypt.edu

Richard L. Blumenthal
Regis University
Denver, CO, USA
rblument@regis.edu

Susan L. Epstein
Hunter College and The Graduate Center of The City University of New York
New York, NY, USA
susan.epstein@hunter.cuny.edu

Monica D. Anderson
University of Alabama,
Tuscaloosa, AL, USA
anderson@cs.ua.edu

DOI: 10.1145/3605215 Copyright held by authors. Publication rights licensed to ACM.

Association for Computing Machinery

Career & Job Center

The #1 Career Destination to Find Computing Jobs.

Connecting you with top industry employers.

Check out these new features to help you find your next computing job.

- Access to new and exclusive career resources, articles, job searching tips and tools.
- Gain insights and detailed data on the computing industry, including salary, job outlook, 'day in the life' videos, education, and more with our new Career Insights.
- Receive the latest jobs delivered straight to your inbox with **new exclusive Job Flash™ emails**.
- Get a free resume review from an expert writer listing your strengths, weaknesses, and suggestions to give you the best chance of landing an interview.
- Receive an alert every time a job becomes available that matches your personal profile, skills, interests, and preferred location(s).

Visit https://jobs.acm.org/