8th International Conference on Higher Education Advances (HEAd'22)
Universitat Politècnica de València, València, 2022
DOI: http://dx.doi.org/10.4995/HEAd22.2022.14217

Challenges of Knowledge Component Modeling: A Software Engineering Case Study

Nikola Luburić, Balša Šarenac, Luka Dorić, Dragan Vidaković, Katarina-Glorija Grujić, Aleksandar Kovačević, Simona Prokić
Department of Computing and Control Engineering, Faculty of Technical Sciences, University of Novi Sad, Serbia.

Abstract

To improve instruction, educators require greater visibility into the learning gains of individual students and the ability to adapt to resolve any learning gaps. A prerequisite to achieving such instruction is decomposing knowledge into small components that function as gradual steps for the learner's journey. However, decomposing a knowledge domain is not trivial, as instructors must overcome many practical challenges along the way.

We report on our experience of knowledge component modeling for the clean code analysis and refactoring domain. We describe our method and list four challenges encountered during modeling and our solution for them. The resulting knowledge component inventory is a secondary contribution of the paper. These results can assist educators in planning and executing knowledge component modeling to refine their instruction and produce more significant learning gains in their students.

Keywords: *knowledge component; intelligent tutoring systems; KLI; cognitive task analysis; clean code; refactoring.*

1. Introduction

Ideal instruction causes robust learning (Koedinger et al., 2012) for the whole audience, with the minimum required time, while engaging and motivating each student to continue learning (Honebein & Reigeluth, 2020). Such a feat seems practically impossible, as most classrooms have dozens or hundreds of students with different prior knowledge, interests, and affective states when the instruction is being delivered (Aleven et al., 2016). This problem is one of visibility and scale (Baker et al., 2022). It is difficult for an educator delivering the lecture to understand the learning gains obtained by an individual student through passive observation. Even with perfect visibility of the students' learning gaps, the educator cannot adapt their instruction for each student, as the number of students far outweighs the number of educators.

Contemporary learning theories coupled with digital technologies provide a scalable way to achieve visibility into individual students' learning gaps and gains (Koedinger et al., 2012; Ritter et al., 2019; Baker et al., 2022). Adaptive e-learning tools, such as intelligent tutoring systems (ITSs), interleave instruction with assessments to ensure a student achieves mastery of a unit (i.e., the expected learning gains) before moving on to the next one. To be effective, ITSs require educators to decompose the domain model (i.e., the taught subject matter) into chunks known as knowledge components (KCs) (Koedinger et al., 2012; Pelánek, 2020).

Even without an ITS, there is value in decomposing a unit into its KCs, communicating them, and monitoring their learning (Koedinger et al., 2012). Educators can focus their instruction and assessments to target one concept at a time by explicitly defining them, reducing students' working memory load (Sweller, 2020). KCs have been used as a basis for systematic methods that refine instruction, leading to more significant learning outcomes (Gusukuma et al., 2018). However, many practical challenges are involved in modeling KCs, some previously reported (Aleven & Koedinger, 2013; Pelánek, 2020).

We conducted KC modeling for an undergraduate software engineering course by following the method proposed by Gusukuma et al. (2018). We utilized the method to develop KCs, instructional, and assessment items for a set of units concerned with clean code analysis and refactoring. In this paper, we report on our experience and identify a set of challenges and our solutions to them as the paper's primary contribution. These results may help educators of all subject matters improve their instruction and overcome the challenges inherent in KC modeling. We also present part of the resulting KC inventory for the clean code analysis and refactoring domains as secondary contributions of the paper that software engineering educators can directly use.

The rest of the paper is organized as follows. Section 2 describes the related work in which we anchor our contribution. Section 3 presents the method we followed while creating our KCs and presents the resulting set of KCs. Section 4 discusses the identified challenges in

KC modeling. Here we outline our solutions to the challenges and present open issues that remain unsolved. Finally, Section 5 concludes the paper and lists ideas for further work.

2. Related Work

Here we briefly discuss the knowledge-learning-instruction framework (KLI) developed by Koedinger et al. (2012), which defines the baseline conceptual model of our work. We then explore previously defined challenges in KC modeling, which we expand upon in Section 4.

2.1. Knowledge Component Conceptual Model

Koedinger et al. (2012) developed a framework that specifies three taxonomies - kinds of knowledge, learning processes, instructional choices, and the dependencies between them. The framework defines four concepts that interact to cause and assess learning gains:

- Instructional events (IEs) are observable variations in the learning environment that facilitate learning by introducing new knowledge or providing a different perspective on familiar knowledge. Examples of IEs include a statement of fact, an example of a concept, a case study, or an illustrative metaphor.
- Assessment events (AEs) are observable variations in the learning environment that require a submission from the student to infer their mastery of the related knowledge components. Examples of AEs include multiple-response questions, algebra problems, or programming exercises. AEs can be instructional when they provide feedback for submissions, and most ITSs offer such AEs.
- Learning events (LEs) cause unobservable processes that change cognitive and brain states. The student's existing knowledge influences LEs, and LEs create new or refine existing knowledge components.
- Knowledge components (KCs) define an acquired unit of cognitive function or mental structure inferred from performance on a set of related assessment events. The KLI framework defines the KC as a broad term for describing pieces of cognition or knowledge, including facts, concepts, misconceptions, or skills.

KCs vary in granularity and present a hierarchical structure (e.g., to master the broader skill of solving basic arithmetic equations, the student needs to master the addition, subtraction, multiplication, and division KCs). Other relations among KCs exist, such as prerequisite relationships (Pelánek, 2021). When decomposing a unit into its KCs, learning engineers should consider the expected understanding (i.e., KC mastery) that students should have at the start of the unit and build upon that.

Given a set of KCs, each student starts with zero mastery for each KC in the set. When a student makes a submission for an AE, they are graded based on their performance (the correctness of the submission and possibly the time it took them to complete it) (Koedinger

et al., 2012). Tracking a student's mastery reduces over-practice (i.e., when a student masters a KC, additional AEs are redundant) and under-practice (i.e., when a student has not mastered a KC, they need additional IEs and AEs to help them develop the mastery).

2.2. Knowledge Component Modeling Challenges

The fundamental challenge of creating a KC model for a domain is determining which KCs correctly map to the student's cognition, evidenced by the model's ability to accurately predict student performance across tasks and time (Aleven & Koedinger, 2013).

To illustrate, we consider a set of knowledge components related to arithmetic addition: (1) Add two single-digit numbers, where the result does not exceed 10. (2) Add any two single-digit numbers. (3) Add two numbers where the result does not exceed 100. (4) Add any two numbers. (5) Add multiple numbers.

Given that a student achieves mastery for the listed KCs, how can we be sure that they have mastered arithmetic addition in general? How can we ensure that the KC set does not include redundant items (e.g., is the third KC necessary, or does mastery of the second KC guarantee mastery of the third)? Given enough students and AEs, Aleven and Koedinger (2013) discuss combining human expertise and statistical analysis to determine better KC models.

Pelánek (2020) reports a set of challenges for KC modeling related to the practical limitations of the activity. A typical domain model for a course can count hundreds of KCs and dozens of IEs and AEs for each KC. Given the ambiguity of KCs discussed by Aleven and Koedinger (2013), creating, managing, and incrementally improving such a collection of items is a considerable practical challenge. Pelánek (2020) offers a list of recommendations to overcome these challenges, including tips for determining KC granularity, reducing possible relationships between elements (e.g., limiting each IE and AE to a single KC; focusing only on the hierarchy relationship in KCs), and automating the creation of AEs where possible.

3. KC Modeling Case Study

Here we outline our case study, including the method we used to perform KC modeling and the achieved results.

We employed KC modeling for a 3rd-year undergraduate *Software specification and modeling* course. The original course was developed without awareness of the KLI framework. The instruction and assessments have been developed without a formal process, relying on informed experience.

We targeted units related to the topic of clean code analysis and refactoring (Martin, 2009; Fowler, 2018), a set of skills whose goal is to produce higher-quality code, making the software more maintainable. For these units, we employed the method for refining instruction

proposed by Gusukuma et al. (2018) and based on the systematic design of instruction (Dick et al., 2015). Here we describe only the steps of the method relevant for our KC modeling.

3.1. Defining the set of KCs

Following Gusukuma et al. (2018), we *identified a set of instructional goals*, including:

- "Assign meaningful names to identifiers in code" entails analyzing the existing code and new requirements and determining a set of words that appropriately describe the targeted identifier.
- "Create clean functions" entails understanding the various aspects that contribute to a function's cleanliness and how to combine them into the codebase.

Next, we *identified common misconceptions* by examining our interaction with previous students and their coding assignments collected as part of a controlled experiment we conducted (Luburić et al., 2022). This analysis resulted in a list of mislearned skills, which served as valuable input for the next activity.

For *instructional analysis*, we examined each instructional goal and defined the expected students' baseline knowledge. In KLI terms, we declared all KCs for which we expected existing mastery from the students (e.g., either from the previous unit of our course or the previous courses). Next, we performed cognitive task analysis (Clark et al., 2007) by analyzing existing course materials and related literature (e.g., Martin (2009) and Fowler (2018)) and interviewing our subject matter experts. Through these activities, we derived the knowledge and skills we expected students to develop to accomplish the instructional goal. We then mapped the identified misconceptions from the previous steps to the instructional goal. Guided by the revised Bloom's taxonomy (Anderson & Krathwohl, 2001), we developed the initial KC inventory over several iterations of brainstorming and review. While performing instructional analysis, we ran into several challenges and dilemmas, which we elaborate on in Section 4.

3.2. The KC inventory

We list our KC inventory as a secondary contribution of the paper as supplementary material[1]. The table lists KCs for two units from the clean code analysis and refactoring domain, relevant for all major programming paradigms (i.e., procedural, object-oriented, and functional). Following the recommendations made by Pelánek (2020), we simplify the relationships between KCs and only include the hierarchy relationship (denoted by the Parent KC Id column).

[1] Because we plan to further refine our instruction over the coming years, we maintain an online page that hosts the up-to-date KC inventory (https://github.com/Clean-CaDET/tutor/wiki/Sample-Knowledge-Components).

4. Discussion

Here we expand the work presented by Aleven and Koedinger (2013) and Pelánek (2020). Guided by our experience, we define a set of challenges and solutions for KC modeling.

4.1. Discovering unnecessary KCs

Researchers strive to create fine-grained KCs in their experiments to build a perfect knowledge map of a domain. However, expanding the KC inventory increases maintenance efforts for educators (Pelánek, 2020). In practical terms, it is challenging to determine if additional KCs justify their maintenance cost when they increase learning gains for a small percentile of students. Furthermore, requiring students to complete AEs for these extra (and possibly redundant) KCs can result in over-practice, violating the goal of minimizing instructional time (Koedinger et al., 2012; Honebein & Reigeluth, 2020).

After our brainstorming sessions, we produced KCs that we subsequently excluded from the final inventory. We reviewed each KC and removed it from the inventory if neither of the following conditions was met:

- The students' prior coding assignments contained errors that resulted from mislearning the examined KC, signaling that the KC is important and should be explicitly addressed.
- Our course materials had segments that directly targeted the examined KC, indicating that the students might have achieved mastery because of these materials.

Through this approach, we removed two KCs from the inventory and integrated their knowledge into the higher-level KC (N01 and F03 in the supplementary material).

4.2. Discovering hidden KCs

Some KCs, especially those at a higher level of granularity (Pelánek, 2020), are simple to identify. Basic intuition coupled with a few brainstorming sessions can reveal many skills and subskills for the initial KC inventory. However, many non-intuitive KCs often remain hidden, including integrative KCs (Koedinger et al., 2012) and other non-obvious skills.

Identifying misconceptions helps discover hidden KCs (Gusukuma et al., 2018). We supplemented this approach by examining coarse-grained KCs for which the students' have trouble achieving mastery (i.e., many students required many AEs before reaching the mastery threshold). Once we determined problematic KCs, we applied cognitive task analysis (Clark et al., 2007) to discover three additional KCs (i.e., N04, F04, and F07 in the supplementary material).

Notably, it is not essential to identify all hidden KCs. As discussed in Section 4.1, this can lead to maintainability and over-practice issues.

4.3. Discovering insufficient AEs

AEs help determine the student's KC mastery, provided they properly target the KC. Koedinger et al., 2012 identified the issue of AEs that do not examine if the KC is developed at the appropriate level of abstraction (and is instead overspecialized). Pelánek (2020) recommends creating enough AEs for each KC (i.e., several dozen) to combat this issue.

We found that these guidelines require a significant upfront investment that is wasted if the issues described in Section 4.1 or 4.2 occur (i.e., discarding a KC might discard most of its AEs). Instead, we created five to eight AEs of varying difficulty for each KC. Our goal is to expand the AE set when we encounter overspecialization in students' thinking or discover problematic AEs (e.g., an AE for which most students make an incorrect first submission).

4.4. Discovering poor IEs

An adequately defined KC might have AEs that correctly assess the students' mastery. However, without clear feedback and instructional resources to examine, students might continuously fail to achieve the required level of mastery. Therefore, another issue might arise from poorly constructed IEs that do not produce the expected LEs.

To mitigate this issue, we designed a feedback loop where students can grade and comment on the quality of our instructional materials. Considering that KCs are small chunks of knowledge, we require frequent feedback from the students and therefore design the feedback control to allow swift input from the student (i.e., grade IEs for a KC on a scale of 1 to 3).

5. Conclusion

To improve instruction, educators require greater visibility into the learning gains of individual students and the ability to adapt to resolve any learning gaps. A prerequisite to achieving such instruction is decomposing knowledge into small KCs that function as gradual steps for the learner's journey. Each step (i.e., KC) is supported by IEs that overcome learning gaps and AEs that test learning gains.

Decomposing a knowledge domain into KCs is not trivial, as instructors must overcome many practical challenges along the way. In this paper, we report on our experience of KC modeling for the clean code analysis and refactoring domain. We describe our method and list four challenges for KC modeling and our solution to them, expanding on previous work in the field. The results can assist educators in refining their instruction and producing more significant learning gains in their students. We provide a KC inventory as a secondary contribution, useful to software engineering educators. Further work entails performing KC modeling for course units and refining the KC, IE, and AE inventory over the coming years.

References

Aleven, V., & Koedinger, K. R. (2013). Knowledge component (KC) approaches to learner modeling. *Design Recommendations for Intelligent Tutoring Systems*, 1, 165-182.

Aleven, V., McLaughlin, E. A., Glenn, R. A., & Koedinger, K. R. (2016). Instruction based on adaptive learning technologies. *Handbook of research on learning and instruction*, 2, 522-560.

Anderson, L. W., & Krathwohl, D. R. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Longman.

Baker, R. S., Boser, U., & Snow, E. L. (2022). Learning Engineering: A View on Where the Field is at, Where it's Going, and the Research Needed.

Clark, R. E., Feldon, D., van Merriënboer, J., Yates, K, and Early, S. (2007). Cognitive Task Analysis. In Spector, J. M., Merrill, M. D., van Merriënboer, J. J. G., & Driscoll, M. P. (Eds.) *Handbook of research on educational communications and technology (3rd ed.)*. Mahwah, NJ: Lawrence Erlbaum Associates.

Dick, W., Carey, L. & Carey, J.O. (2015). *The Systematic Design of Instruction*. Boston, MA: Pearson.

Fowler, M. (2018). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

Gusukuma, L., Bart, A. C., Kafura, D., Ernst, J., & Cennamo, K. (2018, February). Instructional design+ knowledge components: A systematic method for refining instruction. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 338-343).

Honebein, P. C., & Reigeluth, C. M. (2020). The instructional theory framework appears lost. Isn't it time we find it again?. *Revista de Educación a Distancia*, 64(20), 1-24.

Koedinger, K. R., Corbett, A. T., & Perfetti, C. (2012). The Knowledge-Learning-Instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive science*, 36(5), 757-798.

Luburić, N., Vidaković, D., Slivka, J., Prokić, S., Grujić, K. G., Kovačević, A., & Sladić, G. (2022). Clean Code Tutoring: Makings of a Foundation. In *CSEDU*. In press.

Martin, R. C. (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

Pelánek, R. (2020). Managing items and knowledge components: domain modeling in practice. *Educational Technology Research and Development*, 68(1), 529-550.

Pelánek, R., 2021. Adaptive, Intelligent, and Personalized: Navigating the Terminological Maze Behind Educational Technology. *International Journal of Artificial Intelligence in Education*, pp.1-23.

Ritter, F. E., Tehranchi, F., & Oury, J. D. (2019). ACT-R: A cognitive architecture for modeling cognition. *Wiley Interdisciplinary Reviews: Cognitive Science*, 10(3), e1488.

Sweller, J. (2020). Cognitive load theory and educational technology. *Educational Technology Research and Development*, 68(1), 1-16.