# Future Skills: How to strengthen computational thinking in all software project roles

Gudrun Socher*, Sarah Ottinger*, Veronika Thurner*, Ralph Berchtenbreiter[o]
*Department of Computer Science and Mathematics | [o]Department of Tourism
Munich University of Applied Sciences, <firstname>.<lastname>@hm.edu



## Abstract

The digital transformation leads to software systems pervading almost all spheres of private and professional life. To ensure that these software systems are designed and successfully implemented as needed, intensive collaboration is essential in the key roles in software projects, in particular for the roles of product owner, user experience designer, as well as software engineer. The collaboration of people with usually different levels of IT-savviness requires the appropriate skills of those involved, which are also called Future Skills. Computational thinking is an important skill for everyone involved in software projects, no matter which role they are in.

We describe an interdisciplinary tool-based teaching-and-learning program where we build virtual voice-based assistants (voice apps for Amazon Alexa) in interdisciplinary student teams to train computational thinking and collaboration skills. A first competency test validates the effectiveness of our approach.

## Motivation: Future Skills[1]

In current digitalization initiatives, there is a lot of discussion on how to increase graduation numbers in software engineering related study programs in order to have more skilled people driving the ongoing digital transformation. In this discussion, however, we often forget that digitalization is always related to an application domain. The digital transformation benefits strongly if software-related skills are strengthened not only for the core software engineering roles, but also for less-technical roles in software projects like product owners, or user experience designers (see Figure 1), as well as for even more general roles such as product management or project management.

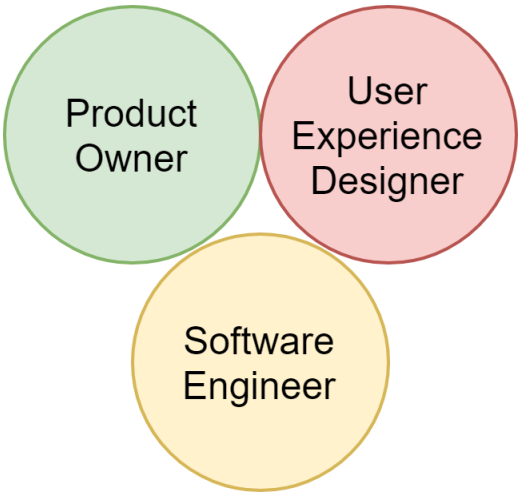


Figure 1: Key roles in software projects.

Software products can only be successful if the key roles work hand in hand in software projects: product owners with their knowledge of the application domain and product vision, user experience designers who guide human-computer interaction, and software engineers being responsible for software implementation. The skills and competencies related to these roles are essential in successful projects. They are required to successfully meet the challenges of digital transformation.

How do we best train the talents for the ongoing digital transformation, and what exactly do future employees need to learn? Stifterverband, a joint initiative of German organizations, has published a discussion paper in September 2018 together with McKinsey & Company where they address three core competency

[1] With 'Future Skills' we refer to 'competencies' required by university graduates across all majors in the coming years. These competencies are necessary to meet digital requirements as they are currently expected in business and society (cf. (Kirchherr et al., 2018)).

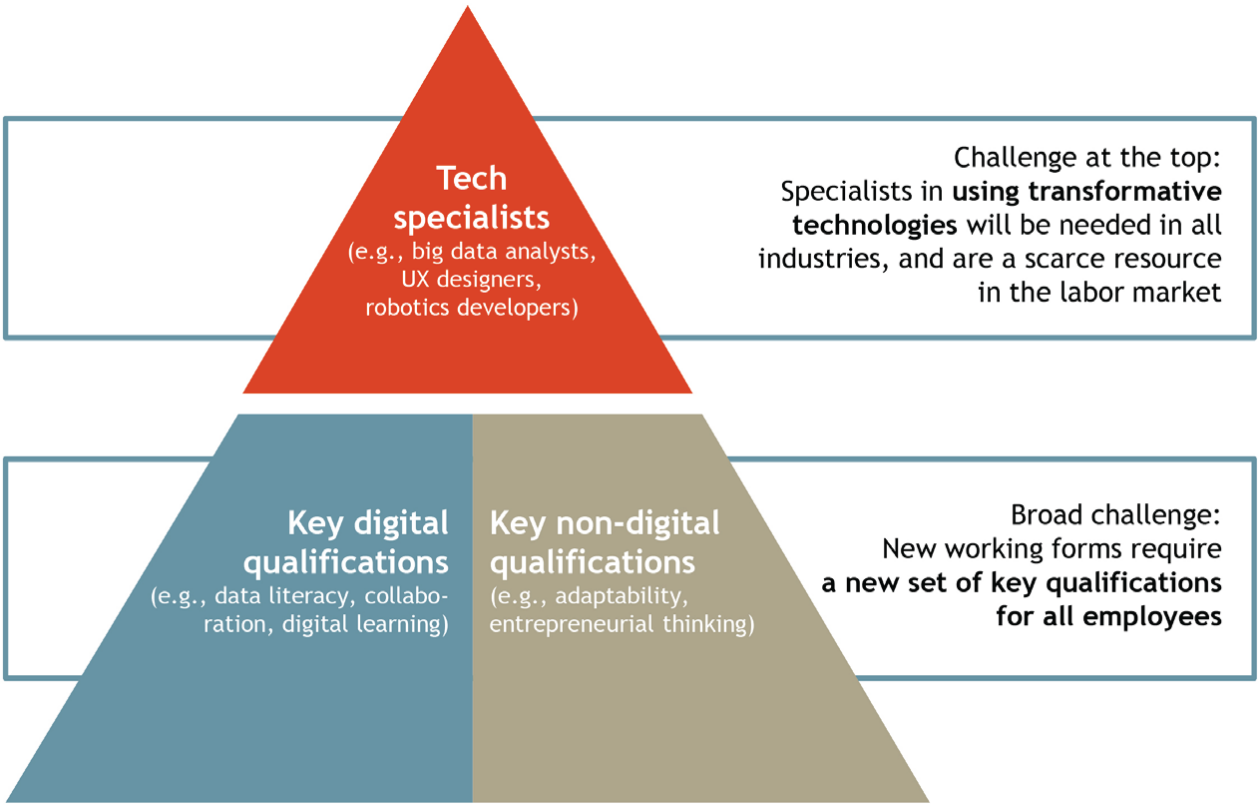


Figure 2: Future Skills: Digitalization and new job profiles lead to two challenges for companies. (1) Jobs shift towards IT related profiles. (2) Work processes and work requirements change for the majority of employees. Many employees thus need a modified set of digital and non-digital key competencies (cf. (Kirchherr et al., 2018)).

categories driving the digitalization (Kirchherr et al., 2018). Figure 2 illustrates competencies related to the digital transformation as well as to new forms of work.

With changes in the job portfolios and new forms of work, there has been an expected shortage of qualified people for some time now. In particular, the job market will be increasingly dominated by job profiles that are heavily related to software engineering. Accordingly, a lot of effort is spent on increasing the number of graduates of software engineering related degree programs, in order to increase the number of software engineers in the market and thus to meet the growing demand for qualified specialists.

At the same time, it is essential to strengthen everybody's digital and non-digital key competencies. Digitalization is not just an IT issue. Digitalization is inherently linked to the digital transformation and thus the creation of digital automation or digital assistance systems in an application domain. Therefore, it is just as important for employees and students of all application domains to acquire software engineering competencies.

Stifterverband (Kirchherr et al., 2018) stresses that both digital and non-digital key competencies are a must-have for all current students. Key digital skills include - among others - data literacy, collaborative skills and digital learning skills. Based on Wing (2006), computational thinking can be considered as a prerequisite for digital learning. By now, the relevance of all these skills for virtually any member of our professional work force is undisputed and widely recognized.

How can we integrate the development of these competencies into study programs? Curricula are often overfull and the amount of available (and often essential) knowledge is increasing in almost every application domain. Solutions have to be found to systematically integrate key competencies required for the digital transformation into the teaching-and-learning processes without weakening the core foundation in the respective application domains.

Project-based learning is a successful instructional learning format that has been proven to systematically strengthen some of the required future skills (Bell, 2010). Interdisciplinary project-based learning is even more effective, as team diversity is an additional success factor for creative, goal-oriented collaboration in addition to the future skills mentioned above (Digitalisierung, 2016; Meier et al., 2007).

## Related work

Wing (2006) introduced the importance of computational thinking for computer science-related tasks 12 years ago. She defines computational thinking as the interplay of decomposition and abstraction and recommends strengthening computational thinking in all study programs.

The ability of abstract thinking, in turn, has long been recognized as a key competency of many technical disciplines, especially for computer science (Bucci et al., 2001; Kramer, 2007) and software engineering (Ghezzi et al., 2002). A distinction is made between static and dynamic abstraction, i.e. the abstraction of structural entities (static) and of processes or behavior (dynamic) (Davis et al., 2014).

The central element of computational thinking is a problem-oriented (as opposed to a solution-oriented) approach (Lorenz and Wurzer, 2014). It is essential to get to the root of a problem or task, to abstract it and to understand contexts and regularities. The goal is to reduce the complexity of the task (keyword: decomposition (Wing, 2006)) and to systematically limit the choice of possible solutions. Only then are potential solution components identified and abstracted into an overall behavior. Computational thinking requires not only the ability to decompose, but also the ability to abstract behavior (Wing, 2006), thus requiring a very high degree of dynamic abstraction in particular. Since algorithms always work on data entities, a corresponding degree of static abstraction is necessary.

In teaching-and-learning practice, it can be observed that not all students have a sufficient level of abstraction and computational thinking to be able to cope with the study program requirements. This is especially true for students of subjects related to computer science. Accordingly, various approaches have been developed to systematically strengthen these abilities (Hazzan and Kramer, 2007; Böttcher et al., 2016). These approaches mainly focus on promoting computational thinking in students of computer science related majors, but do not provide teaching-and-learning concepts for strengthening these skills in students of non-technical subjects, where little to no IT-affinity can be expected.

## Goals

In this work, we develop and apply a teaching-and-learning program for promoting computational thinking in students – and, as an essential basis for this, for fostering students' static and dynamic abstraction competency. To this end, we develop a teaching-and-learning program for interdisciplinary, project-based learning that addresses computer science students as well as non-IT students. Our concept takes into account our students' prior knowledge as well as their individual learning requirements.

In our teaching-and-learning program, we create an easy introduction to digital projects for interdisciplinary student teams. To achieve this, we use a tool-chain for creating voice-based virtual assistants (such as Amazon Alexa), as well as by using Github. Over the last years, the usability of many tools for creating software systems has evolved and improved significantly. Digital tools in the context of software engineering are nowadays no longer editors that are specifically tailored to computer nerds. Rather, nowadays they often are web-based interactive tools that are fun to use and that guarantee good results even for people without IT- and software engineering skills.

Our student teams include tourism majors and computer science students. Github is used as a source code repository by computer science students, as well as a ticketing and project communication tool for all students. We furthermore use the Github project board as a virtual agile board. In this way, we enable all students to make an active, creative contribution in a digital software project even without programming knowledge.

The non-IT students (i.e. students of an application domain) are either in the role of *product owner* or *user experience designer*. The role of *product owner* includes gathering and aggregating the users' needs and desires which requires static abstraction capabilities. The *user experience designer* role focuses on reflecting what users expect. This definitely requires consideration of dynamic processes. In parallel, computer science students deepen their experience in the role of *software engineer* by using new technologies for the implementation of voice-based assistants.

We use pseudonymized pre- and post-tests to analyze to what extent our interdisciplinary tool-based teaching-and-learning program actually fosters the addressed competencies of static and dynamic abstraction in the students.

## Computational thinking and voice-based assistants

Voice-based virtual assistants (voice apps) are currently being massively pushed by major software companies. Examples are Amazon Alexa, Google Assistant, Cortana, and many more. In particular, Amazon and Google provide well-designed, web-based tools that developers can use to create new voice apps with their cloud offerings. These tools and cloud offerings are available free of charge for educational use. In addition, the tools are so sophisticated that it's fun to play with them. In just a few minutes, cool voice-user-interfaces can be created without previous knowledge, so first success can be achieved quickly. Figure 3 shows the Alexa Developer Console, a tool for creating voice apps for Amazon Alexa.

To develop an application for voice-based assistants, the development team must design the Voice User Interface (Voice UI) and implement the business logic. A voice UI must be structured in such a way that the dialogue appears natural to the users. At the

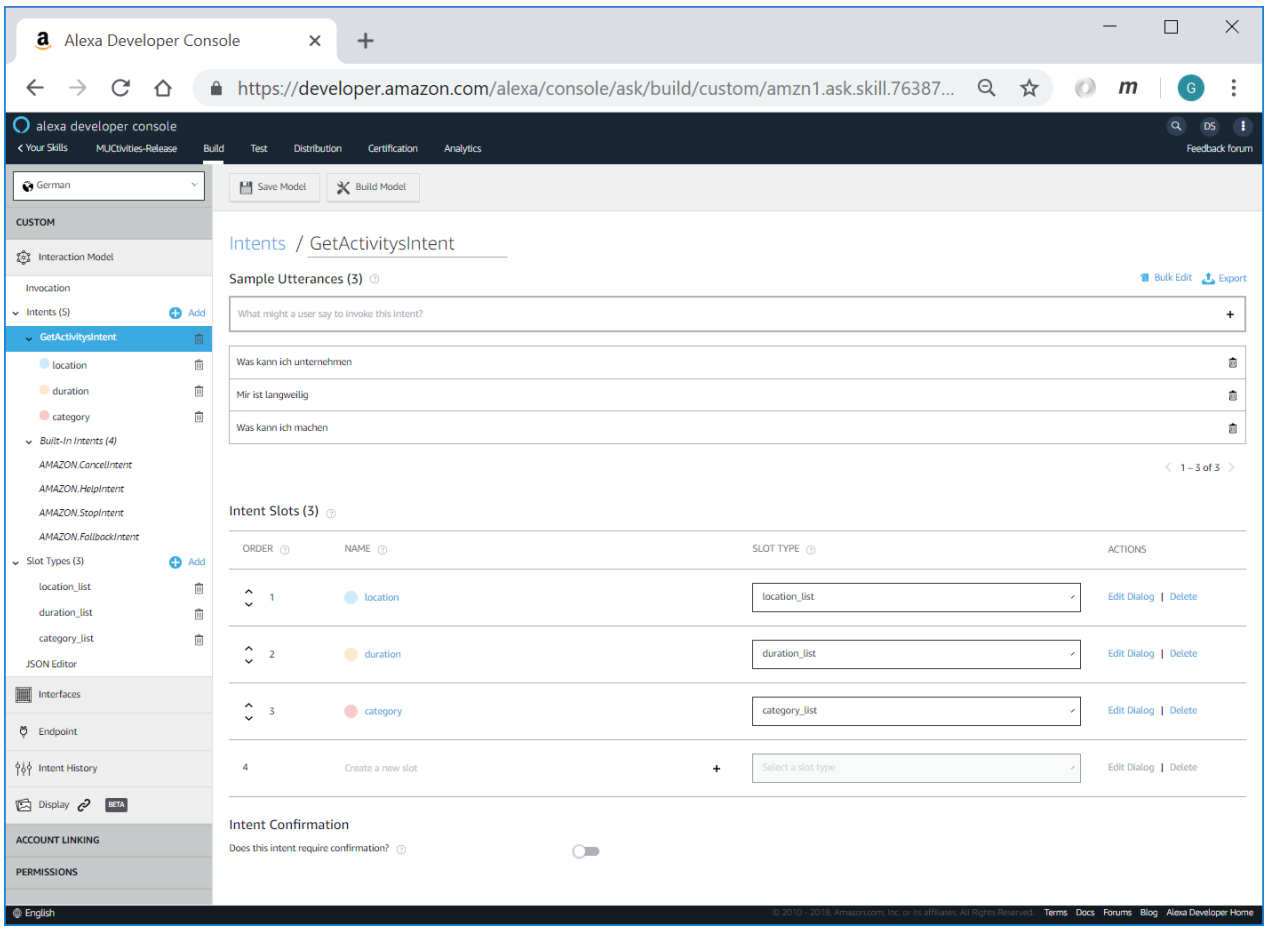


Figure 3: Alexa Developer Console: Web-based tool to create voice apps for Amazon Alexa.

same time, the dialogue must be designed so that the goals expressed by the users or the intentions behind them can be clearly identified and assigned to the implemented business logic. In the jargon of voice UIs, these intentions are called *intents*. Each intent must clearly invoke a feature of the implemented business logic.

Figure 4 shows an example dialog with Alexa for an application called the "joke-of-the-day". A person starts the voice app and gets told one joke. More jokes can be requested. If no more joke is desired, the voice app closes.

Structuring a dialogue into intents strengthens both static and dynamic abstraction abilities.

The identification of the individual intents in a dialogue requires a static abstraction. For example, the joke-of-the-day application is structured into the following intents and the resulting dialog steps:

- welcome
- joke
- closing

The individual dialog steps or intents must then be arranged in a meaningful sequence (the dynamic behavior). This process is rather simple for the joke-of-the-day application (see Figure 5).

The design of a dialog for a voice-based virtual assistant is thus well suited to train both static and dynamic abstraction skills, and to guide students towards computational thinking. Therefore, we use dialog design and implementation for a voice-based assistant as a task for an interdisciplinary tool-based teaching-and-learning program to strengthen computational thinking.

## Interdisciplinary project to strengthen computational thinking

Our didactic concept for the promotion of computational thinking and communicative future skills structures the learning process into several phases (see Table 1). Initially, the students are taught core concepts in non-interdisciplinary groups.

More precisely, the computer science students first learn the technical basics of voice-based assistants, and are introduced to tools for designing and implementing them. Furthermore, computer science students learn requirements engineering. Having that down their belts, these students are well equipped to

| **Task** | **Computer Science Students** | **Students of Application Domain** |
|---|---|---|
| Introduction to voice-based assistants | Examples and tutorial for creating a first voice app | Examples and simulation of dialogues between two partners |
| Idea for voice app and its structure | Invocation, intents, slots | Generating ideas and defining MVP |
| Concept for Voice App | Requirements engineering | Specification of a first dialogue |
| Using tools | Alexa Developer Console, Github | Invocable, Alexa Developer Console, Github |
| 1st Pitch: Students of application domain pitch their ideas to find computer science students for their developer team. | | |
| 1st Sprint | 1st Release | Refining the dialogues |
| 2nd Sprint | 2nd Release | Test 1st Release → Change Requests |
| 2nd Pitch: Voice app demos by computer science students. | | |
| 3rd Sprint | 3rd Release | Test 2nd Release → Final changes and improvements |
| 3rd Pitch: Final presentation with guests. | | |

Table 1: Structure of the teaching-and-learning process throughout the semester for our interdisciplinary tool-based teaching-and-learning program for developing Alexa voice apps.

take on the role of *software engineer* in the interdisciplinary teams that are formed later on.

The task of the application domain students is to develop an idea for a voice-based assistant (in this case an Alexa voice app). For this idea, they then define the dialogue between the user and the voice-based assistant required for a Minimum Viable Product (MVP). This dialog thus contains at least those steps and procedures that are necessary for the minimal functional implementation of the idea. It is important to break down the dialogue into short, simple steps and to structure it. The complexity of creating the voice app is significantly greater than the example of the "joke-of-the-day" shown in Figure 4. Suitable ideas for voice apps are (quiz) games, guides, or useful assistants.

The dialogue is tested and tuned by the students of the application domain. Then, using Invocable[2], the students of the application domain create a first interactive but hard-coded prototype of the voice app.

The interdisciplinary collaboration in the teams begins when the students of the application domain present their ideas to the computer science students. The students of the application domain pitch the prototypes of their voice-based assistants to computer science students and try to motivate them to join their development team. Following these pitches, mixed teams are formed each consisting of computer science students and students of the application domain.

Within the interdisciplinary teams, the computer science students give feedback to the students of the application domain on the design of the dialog that underlies the respective prototype. Based on this, the voice user interface is subsequently refined together. The computer science students take their "natural role" as *software engineer* in the interdisciplinary project. The application domain students, on the other hand, are both *product owner* and *user experience designer*. So they design the interaction between the user and the voice-based assistant in such a way that this interaction is technically meaningful and needs-based from their own perspective. So far in our didactic concept, user experience design is not explicitly taught due to lack of time. However, it would be desirable to improve this in the future.

Computer science students implement the back-end, while students of the application domain are responsible for the front-end in the implementation phase of the voice-based assistant. A simple form of Scrum is useful for organizing the development process into sprints, thus structuring the semester process. Github repositories, including the integrated project boards and the integrated ticketing system (Github issues),

[2]Invocable: `https://www.invocable.com`

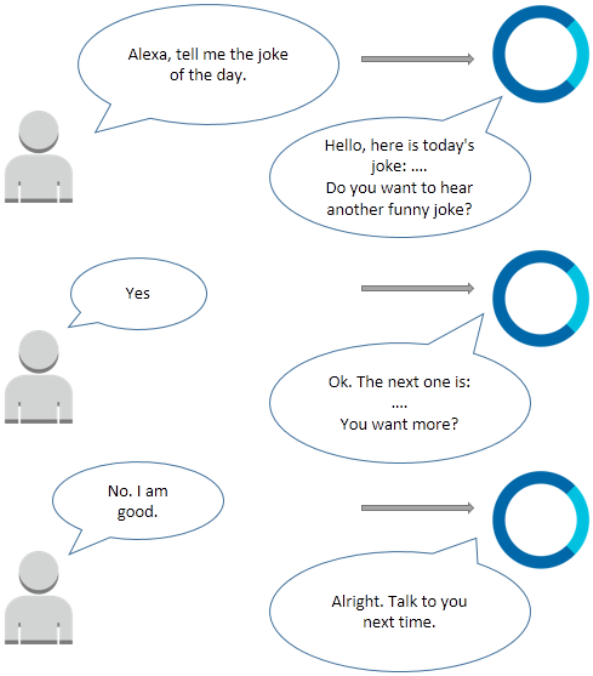


Figure 4: An example dialogue with a voice-based virtual assistant telling jokes.

support team collaboration through an appropriate tooling infrastructure.

This teaching-and-learning program was used for the first time in the winter semester 2018/19 at Munich University of Applied Sciences. Four computer science students (3rd semester) and two to three students of tourism management (6th semester) form a mixed team for the pilot run of our program.

Even if static and dynamic abstraction are not explicitly addressed, all students in this one-semester interdisciplinary tool-based program train their static and dynamic abstraction abilities by structuring and specifying the dialogue between a person and the voice-based assistant in such a way that a corresponding Alexa voice app can be built. All students (including application domain students) work with the Alexa Developer Console and Invocable tools. The use of tools enables all students to work with a working interactive voice app. The working prototype provides rapid feedback so that in particular the students of the application domain can immediately check their dialogue structuring.

From our perspective, this interdisciplinary tool-based teaching-and-learning setting is well suited to foster our students' abstraction skill and more effective in this area than regular class exercises. Furthermore, standardized processes, such as completing Github Issues in team communication, help to structure collaboration.

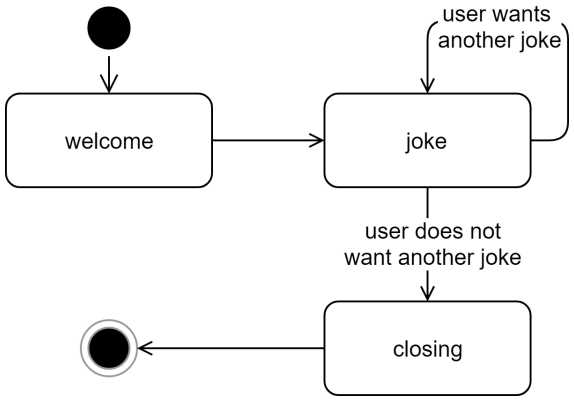


Figure 5: Dynamic structure of the example application joke-of-the-day.

## Assessing computational thinking skills

As we were interested in investigating the development of our students' computational thinking skills during the interdisciplinary tool-based teaching- and learning program, students were requested to work on a competency test at the beginning (week 2) and at the end of the semester (week 11). The test covered two facets of computational thinking, namely static and dynamic abstract thinking processes.

The test was taken by two groups of students, all of which were working on voice-based virtual assistants in a project based way. The first group are computer science students and students of the application domain who worked together in interdisciplinary teams. The other group, our control group, consisted purely of computer science students.

All these students were asked to solve two tasks that both form our competency test and that require static and dynamic abstract thinking processes, respectively. Using pseudonyms allowed us to match the students' pre- and post-tests and thus, to analyze their individual learning outcomes. (Regarding the tourism-management students' test performances, we will discuss the results in the next section "Coding scheme of the abstract thinking test & first results". The test performances of the computer science students have also been analyzed, but will not be presented in this paper.)

Our hypothesis is that the computational thinking skills of students that actively participated in our interdisciplinary tool-based program increased significantly. More specifically, we assume that all students benefit from developing and implementing their innovative ideas on Alexa voice apps, as both the design and the creation of voice-user-interfaces require static and dynamic abstract thinking processes. Note that even though we expect that students working in interdisci-

plinary teams strongly train their collaboration skills during the course, we have not explicitly assessed theses skills in a test.

The first task of our competency test includes a menu of 22 different coffee specialties and requests the students to teach a new barista about the coffee recipes. The menu contains pictures and ingredient lists for each coffee specialty. The challenge is to structure the various coffees and their ingredients in such a way that a new barista can quickly grasp and learn them. In the second task, students are asked to generalize the abstraction process they applied when solving the first task.

To successfully accomplish the second task, the participants first have to become aware of their own actions, identify and structure those processes, and derive a procedure that would be transferable to similar tasks. Therefore, they have to dynamically abstract from their own approaches and accurately document their solutions. From our perspective, task 1 mainly deals with static abstract thinking processes, whereas task 2 covers dynamic abstract thinking processes.

Students are allowed 20 minutes for working on the first task and 15 minutes for completing the second task. Once the students begin working on task two, they are not allowed to use the documents they have generated in task one. In winter semester 2018/2019, 18 tourism students and 54 computer science students participated in the computational thinking competency test.

## Coding scheme of the computational thinking competency test & first results

To analyze the competency test of computational thinking skills, a comprehensive coding scheme was developed incorporating five criteria to evaluate static abstract thinking processes (S1-S5) and three criteria to measure dynamic abstract thinking processes (D1-D3). We defined two additional criteria operationalizing one's ability for self-reflection (R1) and for identifying the requirements needed (R2). All criteria differentiate between four levels of competency (outstanding, good, satisfactory, not yet satisfactory). Score 1 characterizes the lowest level of competency, score 4 the highest one.

We attempted to capture static abstract thinking processes by evaluating students' performances along these four criteria:

- Criterion S1: developing categories that ideally simplify the given representations (of coffee specialties) and structuring ingredients;
  Levels of competency may be described as 'unrefined', 'put together', 'structured' and 'organized' (Hershkowitz et al., 2001).
- Criterion S2: identifying and parameterizing attributes.
- Criterion S3: presenting categories in a structurally-sound way, e. g. as UML-diagrams.
- Criterion S4: using formal notations in a logically consistent way.
- Criterion S5: recognizing familiar structures and realizing that the structures are coherent in a given situation.

Dynamic abstract thinking processes are operationalized by defining these three criteria:

- Criterion D1: identifying a coherent chain of processes.
- Criterion D2: presenting a coherent chain of processes in a structurally sound way.
- Criterion D3: relating the identified chain of processes to one's own solution, e. g. using one's own approach as a basis for generalizing and inferring appropriate processes.

We considered two additional criteria, characterizing one's ability for self-reflection as well as metacognitive thinking skills:

- Criterion R1: identifying errors and reflecting one's own approach.
- Criterion R2: taking the users' preferences into account, as well as the requirements.

First results indicate that the computer science students demonstrate a significantly higher initial level of static abstract thinking skills (mean-value $M=2.04$; standard-derivation $SD=0.49$), in comparison to the tourism majors ($M=1.77$; $SD=0.46$) ($t=2.119$; $df=70$; $p=0.030$).

Regarding the students' initial dynamic abstract thinking skills, we found no statistically significant differences between the performances of computer science students ($M=2.12$; $SD=0.51$) and tourism students ($M=2.28$; $SD=0.43$; whereby $U=384.00$; $p=0.210$). It seems that tourism students have reached a slightly higher level of dynamic abstract thinking skills than computer science students.

Furthermore, the results indicate that computer science students ($M = 2.07$, $SD = 0.78$) and tourism students ($M = 1.81$, $SD = 0.73$) do not differ significantly in their self-reflection skills ($U = 384.00$, $p = 0.210$). We observe only a slight tendency in favor of the computer science students.

Overall, the competency test gives some insight that most of the students – computer science students (3rd semester) as well as tourism students (6th semester) – struggle with demonstrating their static and dynamic abstract thinking skills. According to Wing (2006) and based on our results, we strongly recommend the integration of interdisciplinary teaching- and learning-programs in students' curricula to foster students' computational thinking skills.

Regarding students' reflection skills, it seems to be important to encourage students to reflect their own approaches and to think about the requirements needed.

The results of the pre- and post-tests provide empirical evidence that the tourism students' static abstract thinking skills development can be characterized by a significant increase ($t = -3.986$, $p = 0.001$) supporting our hypothesis. After actively participating in the interdisciplinary tool-based teaching-and-learning program ($M = 2.21$, $SD = 0.48$) the tourism students score significantly better in terms of their static abstract thinking skills than before their participation in this program ($M = 1.77$, $SD = 0.46$). The effect size by Cohen (1992) is around $r = 0.695$, thus indicating a strong effect. Regarding the tourism students' dynamic abstract thinking skills, no significant effect can be reported ($t = -1{,}475$, $p = 0.159$). However, there is also a tendency towards an increase of dynamic abstract thinking skills ($M = 2.24$, $SD = 0.44$). Our hypothesis that tourism students improve their computational thinking skills during the interdisciplinary tool-based program can be confirmed.

## Challenges and experiences

Project-based teaching presents many challenges to teachers (Barron et al., 1998), among others:

- Formulate clear definitions of learning goals and competencies students should acquire.
- Make sure that the project tasks cover the planned learning content to the desired extent and adequately demand and strengthen the competencies to be acquired.
- Build social interaction structures within the project teams that allow a balanced distribution of roles and tasks.
- Create a good (i.e. applicable) schedule.

128 students in computer science and tourism management were in the pilot group in the winter semester 2018/19. We combined a software engineering module and a module for digital marketing and management. Both modules have their own learning objectives and content. In addition to the learning objectives of these respective modules, additional learning objectives of the interdisciplinary tool-based teaching-and-learning program include the increase in static and dynamic abstraction abilities mentioned in this paper as well as the improvement of collaboration and communication skills in digital projects. Therefore, instructors need to encourage students in their learning process. At the same time, instructors have to take care to not overburden their students.

We solve the challenges of the different learning objectives and content of the initial modules by running some part of the interdisciplinary collaboration between the students in a purely virtual way, using the cloud-based Alexa Developer Console as well as Github and Github issues. The classroom events where both computer science students and students of the application domain are together are the three pitches which are highlighted in gray in Table 1, i.e. the pitch of the prototypes by the students of the application domain, the pitch of the voice app demos by the computer science students, and finally the joint pitch during the final presentation. A mix of virtual and physical collaboration creates enough space for both teaching and learning sessions to accommodate the specific contents of the respective modules and the corresponding competencies according to the definition of learning goals.

The interdisciplinary project was a lot of fun for everyone involved. For that reason alone, it is highly recommended to repeat the project. The use of new web-based development tools was well received by all participating students independent of their field of study.

The computer science students were motivated primarily by the fact that new voice technologies were used in the context of this project. In turn, tourism students have grown into the role of *product owner* during the project. Furthermore, by using the development tools, they were in able to increase their competency in using web-based tools. They also liked to creatively integrate sound effects into the Alexa voice apps.

## Summary and outlook

It is important for all students to develop and to strengthen their ability of computational thinking in order to meet the requirements of the digital transformation. As a basis, it is helpful that the students first develop the skills for static and dynamic abstraction as these are a basic building block of the ability of computational thinking. Computational thinking is one of the Future Skills (Kirchherr et al., 2018), which are important core competencies for the future working life as well as for the participation in business and society in the era of the digital transformation and the new forms of work linked to it.

We strengthen computational thinking through interdisciplinary, tool- and project-based learning. As a project topic and work context, we select the design and implementation of voice-based digital assistants (so-called voice apps). The students are encouraged to use static and dynamic abstraction for the specification of the dialogue between a human and a voice-based assistant.

A competency test was developed in order to measure the effectiveness of our approach. The test was run at the beginning and at the end of the semester. (The evaluation of the post-test is not yet completed.) The pseudonymized test results are used to determine the extent to which the students were able to improve

their abilities of static and dynamic abstraction during the program.

More tests are required for a more detailed analysis of future skills through interdisciplinary tool- and project-based learning in digital projects. Accordingly, we plan to improve and further expand our approach so that additional competencies are targeted and the effects are captured by additional measuring instruments.

## References

Barron, B. J., D. L. Schwartz, N. J. Vye, A. Moore, A. Petrosino, L. Zech, and J. D. Bransford
1998. Doing with understanding: Lessons from research on problem-and project-based learning. *Journal of the Learning Sciences*, 7(3-4):271–311.

Bell, S.
2010. Project-based learning for the 21st century: Skills for the future. *The Clearing House*, 83(2):39–43.

Bucci, P., T. Long, and B. Weide
2001. Do we really teach abstraction? In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, New York, USA. ACM.

Böttcher, A., K. Schlierkamp, V. Thurner, and D. Zehetmeier
2016. Teaching abstraction. In *2nd International Conference on Higher Education Advances, HEAd'16*, P. 357–364, València. Universitat Politècnica de València.

Cohen, J.
1992. Statistical power analysis. *SAGE Journals*, 1(3):98–101.

Davis, D., T. Yuen, and M. Berland
2014. Multiple case study of nerd identity in a cs1 class. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education, SIGCS'14*, P. 325–330, New York, USA. ACM.

Digitalisierung, H.
2016. The digital turn – hochschulbildung im digitalen zeitalter. *Edition Stifterverband, Berlin*.

Ghezzi, C., M. Jazayeri, and D. Mandrioli
2002. *Fundamentals of Software Engineering*. Prentice Hall PTR.

Hazzan, O. and J. Kramer
2007. Abstraction in computer science & software engineering: a pedagogical perspective. *Frontier Journal*, 4(1):6–14.

Hershkowitz, R., B. B. Schwarz, and T. Dreyfus
2001. Abstraction in context: Epistemic actions. *Journal for Research in Mathematics Education*, Pp. 195–222.

Kirchherr, J., J. Klier, C. Lehmann-Brauns, and M. Winde
2018. Future Skills: Welche Kompetenzen in Deutschland fehlen.

Kramer, J.
2007. Is abstraction the key to computing. *Communications of the ACM*, 50.

Lorenz, W. E. and G. Wurzer
2014. Algorithmisches denken. In *Brickster Style – Digitales Entwerfen eines Kulturzentrums in Wien-Meidling*, P. 7–10. Technische Universität Wien.

Meier, A., H. Spada, and N. Rummel
2007. A rating scheme for assessing the quality of computer-supported collaboration processes. *International Journal of Computer-Supported Collaborative Learning*, 2(1):63–86.

Wing, J. M.
2006. Computational thinking. *Commun. ACM*, 49(3):33–35.