文章编号：1672-5913(2020)12-0002-09 中图分类号：G642

Fast Trainable Capabilities in Software Engineering–Skill Development in Learning Factories

André Ullrich*, Malte Teichmann, Norbert Gronau

Abstract: The increasing demand for software engineers cannot completely be fulfilled by university education and conventional training approaches due to limited capacities. Accordingly, an alternative approach is necessary where potential software engineers are being educated in software engineering skills using new methods. We suggest micro tasks combined with theoretical lessons to overcome existing skill deficits and acquire fast trainable capabilities. This paper addresses the gap between demand and supply of software engineers by introducing an action-oriented and scenario-based didactical approach, which enables non-computer scientists to code. Therein, the learning content is provided in small tasks and embedded in learning factory scenarios. Therefore, different requirements for software engineers from the market side and from an academic viewpoint are analyzed and synthesized into an integrated, yet condensed skills catalogue. This enables the development of training and education units that focus on the most important skills demanded on the market. To achieve this objective, individual learning scenarios are developed. Of course, proper basic skills in coding cannot be learned over night but software programming is also no sorcery.

Key words: learning factory; programming skills; software engineering; training

1 Introduction

The worldwide demand for Software Engineering (SE) skills-that is understanding core computer science concepts and applying engineering knowledge in a social and economic context for shaping software artefacts[1]- is currently increasing[2]. The annual output of universities and internal training programs of companies, which highly rely on IT skills, are by far not sufficient to fulfil this demand[3]. The situation is further aggravated by the fact that conventional capabilities of a typical undergraduate SE student do not align well with industrial needs[4]. One area where the gap between demand and supply is particularly high is the manufacturing industry, since developments like the 4th Industrial Revolution[5] and Internet-of-things propel specific demand for SE capabilities. However, a university diploma is not for all software engineering capabilities necessary. Especially fast-trainable-capabilities (FTC) can be learned without a time-consuming academic degree.

• André Ullrich, Malte Teichmann and Norbert Gronau are with the Department of Business Informatics, University of Potsdam, Potsdam 14482, Germany. E-mail: aullrich@lswi.de; mteichmann@lswi.de; ngronau@lswi.de.
* To whom correspondence should be addressed.
Manuscript received: 2020-02-01; revised: 2020-04-02; accepted: 2020-09-22

We refer to FTC, in the following, as quickly and little time-consuming trainable programming skills with a user centered development focus (and further IT capabilities). Besides conventional training and education measures, innovative approaches to fulfil the demands for SE skilled employees can be applied. In this contribution, a solution approach to a portion of the demand and supply gap is provided that enables to train workers without or with insufficient SE skills to perform several tasks, for which full-fledged software engineer was needed before. This approach aims at enabling workers to easily undertake first steps into software engineering. To achieve this goal, a procedural model was developed that starts with SE requirement collection from both theory and practice. First, diverse competence models for software engineering were analyzed to identify a model that sets the framework for structuring the FTC, which is systematizing the competences and skills queried in practice. Therefore, job advertisements for software engineers on ten highly visited job portals were queried, using the service ALEXA, and then analyzed, synthesized, and clustered into university level and non-university level skills. The latter are addressed by short tasks, building on a scenario and problem-based didactical approach in a learning factory[6].

The remainder is structured as follows. Skill classification approaches for SE are introduced in Sec. 2. Capability requirements for software engineering are identified in Sec. 3. The didactic concept of our learning factory approach is presented in Sec. 4. Two learning tasks are shown in Sec. 5. Conclusions are drawn Sec. 6.

2 Skill Classification Approaches in Software Engineering

Diverse approaches to structure software engineering skills exist, e.g. according to core principles, essential competences, or diverse skill facets. The software engineering body of skills (SWEBOS) framework focusses on non-technical skills such as collaboration, communication, self-organization, personal competences and problem awareness[7]. Paper[8] differentiate the roles system analyst, software designer, computer programmer and software tester. They emphasize that communication skills, interpersonal skills, organizational skills, analytical and problem-solving skills, being a team player, the ability to work independently and being open and adaptable to changes are the most relevant soft skills for these roles. Paper[9] apply the Critical Incident Interview technique for determining job requirements for software engineers and analyze competences identified by software managers. They derived 38 essential competences for software engineers, structured into the facets interpersonal skills, situational skills, personal attributes and task accomplishment competences, focusing on both soft and technical aspects. Paper[10] differentiates between a body of core computer science concepts, relating e.g. to data structures, algorithms or programming languages, as technical foundation and core content, which is applied through a body of engineering knowledge and complemented by a social and economic context of artefact creation. Paper[1] build on that and emphasize diverse programming languages. The Software Engineering Competency Model (SWECOM) comprises of the elements cognitive skills, behavioral attributes and skills, related disciplines, requisite knowledge, and technical skills. Skill areas for the latter are, inter alia, human-computer interaction, software measurement, software configuration management, software safety and security or software engineering. Paper[11] uncovered in interviews with software engineers 53 attributes of great engineers, structured into the categories personal characteristics, decision making, teammates, and software product. Paper[12] conducted a literature review with a focus on soft skills and identified self-reflection, conflict resolution, communication, and teamwork as top taught skills. The Software Engineering Body of Knowledge (SWEBOK) lists as relevant software

knowledge areas: requirements, design, construction, testing, maintenance, configuration management, quality, engineering tools and methods, engineering process, and engineering management. Knowledge of related disciplines such as systems engineering or computer science is important too[13].

To identify a theoretical fundament for FTC in software engineering and to enable non-software engineers to conduct software engineering tasks on the basis of a didactical approach the classification approaches are analyzed regarding four comparison criteria: (1) differentiation fast vs. long trainable capabilities, (2) differentiation of programming language skills, (3) explicated didactical approach, and (4) user-centered software development focus enables.

While paper[7] and paper[9] fulfil at least criterion (4), the paper[8]. does not meet any criteria. Paper[12] present a didactical approach but do not address another criterion. A specific didactical focus can also be found in the paper[10], which meets all criteria except a clear differentiation of programming language skills. Criterion (2) seems difficult to address: only paper[11] explicitly differentiate between programming languages. On the other hand, they neither distinguish between fast and long trainable capabilities nor present a didactical approach. The same applies to paper[13], who, however, partially address criterion (2). Paper[1] and the IEEE Computer society address criterion (2) partially, too. In other papers, a didactical approach (3) and a user-centered software development focus (4) is presented as well. While fulfils criterion (1) partially, paper[1] present a clear differentiation between fast and long trainable capabilities. Table 1 summarizes the analysis of the skill classification approaches. Against the background of the introduced criteria, paper[1] represent the conceptual basis for structuring the practically demanded software engineering skills as well as enriches the didactical basis of the learning tasks.

Table 1 Comparison of skill classification approaches.

	Differentiation fast vs. long trainable capabilities	Differentiation of programming language skills	Didactical approach explicated	User-centered software development focus
Paper[9]: Identifying Essential Competencies of Software Engineers	No	No	No	Yes
Paper[10]: Software Engineering for the 21st Century: A basis for rethinking the curriculum	Yes	No	Yes	Yes
Paper[8]: Soft Skills and Software Development: A Reflection from the Software Industry	No	No	No	No
IEEE Computer society: Software Engineering Competency Model	Half	Half	Yes	Yes
Paper[7]: SWEBOS – The Software Engineering Body of Skills	No	No	No	Yes
Paper[11]: What Makes A Great Software Engineer	No	Yes	No	Yes
Paper[1]: Half a Century of Software Engineering Education. The CMU Exemplar	Yes	Half	Yes	Yes
Paper[12]: Software Engineering Education Beyond the Technical: A Systematic Literature Review	No	No	Yes	No
Paper[13]: SWEBOK V3.0. Guide to the Software Engineering Body of Knowledge	No	Half	No	Yes

3 Fast Trainable Capabilities for Software Engineering

Considering the gathered data from the job advertisements for software engineers on the job portals as a whole, programming skills are by far the most required skills. Therein, knowledge and the ability to code in object-oriented languages as well as applicable knowledge in data bases are especially in focus. These facets are followed by job experience and soft skills such as communication and self-organization skills. With a large distance, an official degree is required. Furthermore, extra requirements and bonuses such as language skills (e.g., good command of written and spoken English, or Cantonese with the ability to speak Mandarin), or competences related to working with physical components (e.g., troubleshooting of hardware problems) are in demand (Fig. 1).

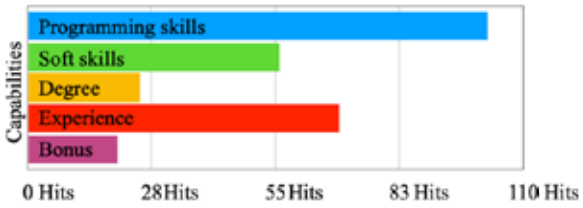

Fig. 1 General requirements on software engineers.

In summary programming skills and soft skills are more valued than official degrees by the market. Hardware related skills and experience are, nonetheless, necessary too. Which specific skills are needed, however, strongly depends on the respective job profile. The identified software related competences often have a lower specific coverage in the profiles.

An in depth consideration of programming skills unveils that knowledge in specific programming languages (32%) such as Python or Ruby and knowledge in central concepts (23%) like data structure, software design patterns or object-oriented programming are the most required hard skills in this category, followed by the capability of working with specific frameworks and tools as well as debugging and troubleshooting and the ability to work with databases. Working with cloud technologies is the least required programming skill. Considering soft skills in more depth, the distribution is rather homogeneous: Communication skills (21%), teamwork (20%), analytical thinking and problem solving (20%), or self-organization (18%) are the most demanded skills. This category is completed by motivation for life-long learning (11%), the ability to work in agile environments (5%), and self-efficacy (5%) (Fig. 2).

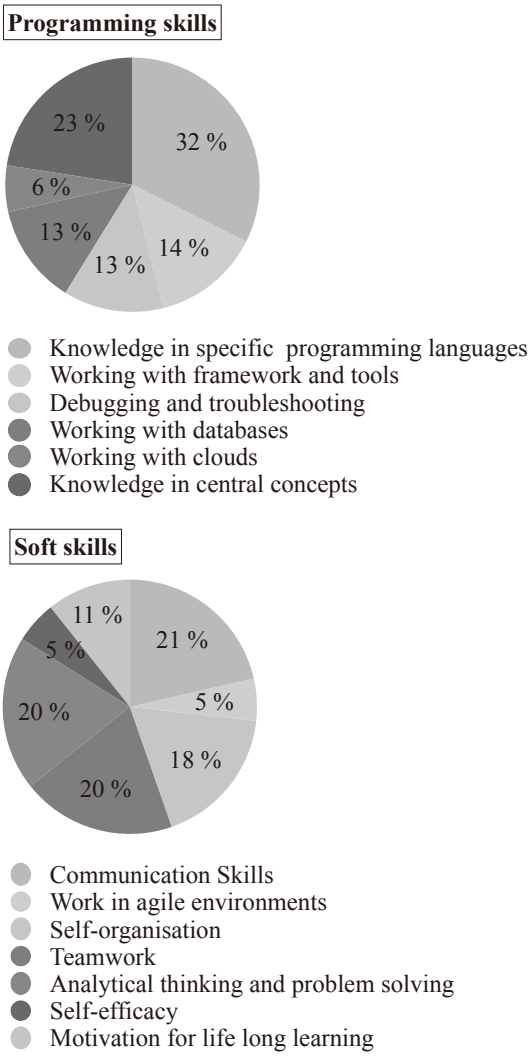

Fig. 2 Distribution of programming and soft skills.

For the sake of focusing on conveyable FTCs, we will concentrate in the following on technical requirements and neglect the other skill facets.

These capabilities can be distinguished into such, which require profound theoretical knowledge (PTK) typically acquired for a university degree and such, which can be trained or learned otherwise, since no need for a deep understanding of, e.g., software architecture principles, or operating systems exists for usual task conduction. From this catalogue of capabilities we isolate these that can be acquired during several one-day-trainings in theoretical and practical micro learning units. In Table 2, programming skills are assigned to these two categories, complemented with a brief description. Learning factories as practical learning environments are ideal places for developing FTC in practical learning tasks, which we will show in the following.

Table 2 Programming skills.

Capabilities	Brief description	FTC	PTK
Knowledge in specific programming languages	Skills in object-oriented programming languages (e.g., Java, Python, C++, C, Ruby, etc.)	X	
Working with framework and tools	Ability to work with current frameworks (Spring, Dropwizard, Spring boot) and tools (especially GIT)	X	
Debugging and trouble-shooting	Ability to analyze and debug for the sake of troubleshooting and problem solving of own-programmed and third party software	X	
Working with databases	Refers to skills in SQL, JSON, MySQL, REST API, or NoSQL	X	
Working with clouds	Knowledge of cloud platforms (e.g., AWS, Cassandra, Kafka, Nutanix), their internal infra-structure and of bridging between them	X	
Knowledge in central concepts	Understand of core computer science concepts, e.g. algorithms, data structure, software design patterns, OOP and knowledge of software engineering processes (e.g. agile, scrum, continuous integration etc.)		X
Computer science fundamentals	Knowledge in computer science fundamentals (project planning, software design/development process etc.)		X

4 Learning Factory in the Research and Application Center Industry 4.0

Learning factories are learning environments in which participants are trained by the usage of simulated real production processes, which are as realistic and authentic as possible[14]. They offer a basis for self-controlled and informal learning and pursue an action-oriented approach for competence development by means of structured self-learning processes that are supported by different teaching methods, which move the teaching and learning processes close to real situations[15]. Digitized learning factories cover topics of digitization, make use of new digital possibilities within their didactic approach, and use "digital tools for the purpose of learning production related concepts and subjects" [16]. That is, relevant aspects of production are hardware-and/or software-based replicated and can be created, tested, and simulated for conveying skills, e.g. in the fields of software engineering[17].

The learning factory in the Research and Application Center Industry 4.0 (RACI 4.0) comprises a hybrid learning environment, which combines the benefits of virtual and hardware simulation in order to design or analyze industrial manufacturing processes or value-adding networks as well teach, learn and understand the advantages of using new technologies in industrial processes. The main physical components are work pieces and machine tool demonstrators as well as transport lines which connect various machine tool demonstrators. Additionally, Internet of Thing (IoT) devices such as AR/VR glasses, tablets, smartwatches, robots, smart products and machines are integral elements. Especially the IoT technologies as well as the therefore required skills for programming and engineering are in theoretical and practical focus

within learning factory sessions.

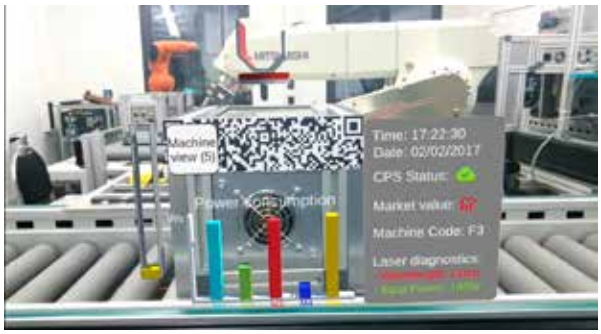

Fig. 3 Example scenario 1: extension of a user interface of AR glasses.

The scalable design of learning modules and learning tasks allows for addressing different skill levels. On the one hand, it is possible to teach the principles of software architecture. On the other hand, programming skills are also trainable by little demanding learning tasks. For the purpose of developing IoT and software engineering skills, a subject-oriented and scenario-based didactic concept has been developed and implemented[18]. The scenarios are either from a repository or a simulation of real tasks which employees have to fulfil (in the future). These scenarios can be merged into a whole teaching and learning environment by connecting different tasks. The learning modules of the didactic concept are content-specific and consist of several learning elements. From a competence facet perspective, a module focusing software engineering skills in the RACI 4.0 learning factory is the module basics in software engineering. In alignment with[1] the foci lie on computer science fundamentals like software architecture, software design principles and engineering quality assurance as well on the conveyance of basic coding skills in practical learning units.

5 Practical Teaching and Learning Tasks

Focusing on conveying and applying programming-related skills, we introduce 2 small tasks:

• Programming of Augmented Reality glasses: New user actions have to be integrated into the user interface of a pair of AR glasses.

• Programming of a Raspberry Pi: When a light beam is interrupted some actions have to be performed.

Both scenarios primarily address and make use of knowledge in programming languages and working with framework/tools.

5.1 Scenario 1: Extension of a user interface of AR glasses

This scenario is based on paper[19]. The requirement is to add a menu entry which allows for a decision between two alternatives when a certain product is being watched through the AR glasses. The result of the users action shall be shown in the AR glasses (Fig. 3).

```
text_statement_product_is_ok(
   if (text_statement_product_is_ok_transparent){ % yes button is pressed
         visual = true
   }
   if (text_statement_product_is_ok_nontransparent){  % nothing was pressed
         visual = false
   }
)
text_statement_product_is_not_ok(
   if (text_statement_product_is_not_ok_transparent){  % no button is pressed
         visual = true
   }
   if (text_statement_product_is_not_ok_transparent){ % nothing was pressed
         visual = false
   }
)
```

Fig. 4 Partial source code visualization.

The expected course of action of the participants is as follows: With a graphical tool the two buttons “yes” and “no” are to be drawn. Alternatively, some existing button shapes could be dragged into the programming pane and renamed. The reference object and the target image are to be captured and positioned spatially. The buttons have to be positioned in the three-dimensional space, in spatial relation to a reference object, in this case a target image. As a next step, the question has to be answered when the action buttons are to be presented to the user of the AR glasses. This should take place when the AR glasses recognize the target image.

Now the programmer has to describe the behavior of the spatial objects. For instance the pseudo-code for this simple situation is (target image is

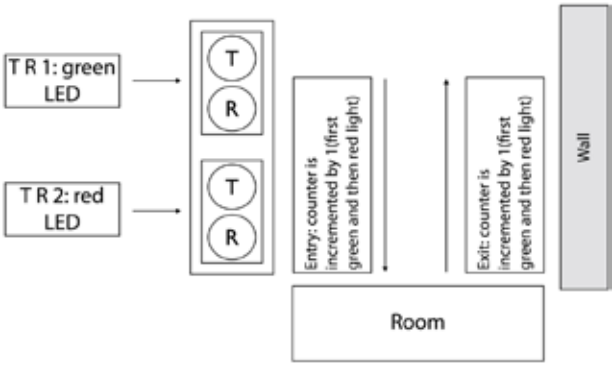

Fig. 5 Conceptual model "Setting of the visitor counter".

recognised [show buttons]) and IF (target image is NOT recognised [hide buttons]). Another example is to formulate environment conditions like "button is pressed" (button. pressed). The visual element "product ok" or "product not ok" will only be visible when some conditions are fulfilled as shown in Fig. 4. The programming of AR interaction is very similar to traditional programming with the addition of a spatial behavior perspective which has to be taken into account. The advantage of AR is that visible and touchable objects are programmed which helps to understand the basic concepts of environment status, conditions and user triggered decisions.

5.2 Scenario 2: Programming a Raspberry Pi

The task in this second scenario is to implement a visitor counter for determining the number of people who are in a room using two distance sensors (Fig. 5). The room has only one entry and only one exit and the respective directions for entering and leaving the room are determined. The room is empty at the beginning and each person that is entering or leaving the room passes a sensor respectively.

At the beginning of this scenario the participants will be provided with a kit of electronic components (raspberry Pi, sensors etc) and a pre-defined program to read a sensor unit.

First, the participants get a short introduction into Python (20 min). In subsequent tasks the pre-defined script has to be extended to fulfill the requirement of counting visitors. To achieve the given requirements the participants draft a conceptual model of the necessary steps and realize them, using Python with

```
# ---------------------------------------------
# Start Python-script
# ---------------------------------------------
os.system('clear')
print „visitor counter"
# GPIO set trigger low, Ausis
GPIO.output(GPIO_TRIG, False)
GPIO.output(GPIO_TRIG2, False)

visitor = 0                                  # with 0 intialising - start value
distancetowall = distancemeasuring1() # distance sensor to wall
halfdistance = distancetowall/2 # differentiation between entry and exit

try:
while True:
    distance1 = distancemeasuring1() # distance measuring Sensor1
    distanc2 = distanzmessung2() # distance measuring Sensor2
    # visitor triggers Sensor1 --> there is one more visitor
    if(GPIO.input(GPIO_ECHO)==1 and GPIO.input(GPIO_ECHO2)==0 and distance1 <
halfdistance):
      visitor = visitor + 1  # add 1 to counter
      time.sleep(2) # 2 seconds time. until the next measurement
    # visitor triggers Sensor2 --> one visitor left
    elif(GPIO.input(GPIO_ECHO)==0 and GPIO.input(GPIO_ECHO2)==1 and distance2 >
halfdistance):
      visitor = visitor - 1   # counter if a visitor leaves
      time.sleep(2)
    # trigger both sensors. one is coming and one is leaving. value counter value stays
the same.
    elif(GPIO.input(GPIO_ECHO)==1 and GPIO.input(GPIO_ECHO2)==1):
      time.sleep(2)
    # in case non visitor is coming/leaving
    elif(GPIO.input(GPIO_ECHO)==0 and GPIO.input(GPIO_ECHO2)==0):
      time.sleep(2)
    # failure prevention, if the Counter values become negative
    elif visitor < 0
      visitor = 0
    return visitor
```

Fig. 6 Realized script in scenario 2.

the assistance of a learning companion. Afterwards the different realized results will be discussed. This allows for reflection of the content and leads to better internalization of the newly acquired skills. The realized script is shown in Fig. 6.

6 Conclusions

In this contribution fast trainable capabilities in software engineering are deduced from theory and practice. These contents are systematized in processable chunks of quickly learnable programming skills. On that basis and applying an action and subject-oriented approach, teaching and learning tasks structured into two scenarios are used to illustrate the capability conveyance in a learning factory. Current activities focus the continuous evaluation, reflection, and improvement of the teaching and learning formats. Further tasks will be the development of additional scenarios as well as extending the theoretical content for the participants. Future work will address strengthening the theoretical foundation, that is extending both content and teaching methods as well as developing further learning tasks and within this modules and scenarios.

Acknowledgement

This work was supported in part by the Junior Research Group “ProMUT” (01UU1705B) which is funded by the German Federal Ministry of Education and Research as part of its funding initiative “Social-Ecological Research” and the German Internet Institute (16DII116).

References

[1] Mead N R, Garlan D, Shaw M. Half a century of software engineering education: The CMU exemplar [J]. IEEE Software, 2018, 35(5): 25-31.

[2] Aasheim C L, Li L, Williams S. Knowledge and skill requirements for entry-level information technology workers: A comparison of industry and academia [J]. Journal of Information Systems Education, 2019, 20(3): 10.

[3] Tuzun E, Erdogmus H, Ozbilgin I G. Are computer science and engineering graduates ready for the software industry? Experiences from an industrial student training program[C]// Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET ’ 18). New York: ACM, 2018: 68-77.

[4] Heggen S, Cody M. Hiring millennial students as software engineers: a study in developing self-confidence and marketable skills[C]//Proceedings of the International Workshop on Software Engineering Education for Millennials (SEEM). Gothenburg: IEEE/ACM, 2018: 32-39.

[5] Beier G, Ullrich A, Niehoff S, et al. Industry 4.0: How it is defined from a sociotechnical perspective and how much sustainability it includes–a literature review[J]. Journal of Cleaner Production, 2020, 259: 1-13.

[6] Teichmann M, Ullrich A, Gronau N. Subject-oriented learning – a new perspective for vocational training in learning factories [J]. Procedia Manufacturing, 2019, 31: 72-78.

[7] Sedelmaier Y, Landes D. Swebos–the software engineering body of skills [J]. International Journal of Engineering Pedagogy, 2015, 5(1): 20-26.

[8] Ahmed F, Capretz L F, Bouktif S, et al. Soft skills and software development: A reflection from the software industry[J]. International Journal of Information Processing and Management, 2013, 4(3): 171 - 191.

[9] Turley R T, Bieman J M. Identifying essential competencies of software engineers[C]// Proceedings of the 22nd Annual ACM Computer Science Conference on Scaling up : Meeting the Challenge of Complexity in Real-world Computing Applications. New York: ACM, 1994: 271-278.

[10] Shaw M. Software Engineering for the 21st Century: A basis for rethinking the curriculum[R]. Carnegie Mellon University: Technical Report CMU-ISRI-05-108, 2005.

[11] Li P L, Ko A J, Zhu J. What makes a great software engineer?[C]// Proceedings of the 37th International Conference on Software Engineering. Piscataway: IEEE , 2015: 700-710.

[12] Groeneveld W, Vennekens J, Aerts K. Software engineering education beyond the technical: A systematic literature review[C]// Proceedings of the 47th SEFI Conference. Brussels : SEFI – European Society for Engineering Education, 2019: 1607-1622.

[13] Bourque P, Fairley R E. SWEBOK V3.0. Guide to the Software Engineering Body of Knowledge [M]. Washington D.C.: IEEE Computer Society Press, 2014.

[14] Abele E, Eichhorn N. Process learning factory – Training students and management for excellent production processes[C]// Proceedings of the Kuljanic, E. (Ed.) Advanced Manufacturing Systems and Technology. Udine: CISM, 2008: 63-73.

[15] Abele E, Metternich J, Tisch M, et al. Learning factories for research, education, and training[J]. Procedia CIRP, 2015, 32: 1-6.

[16] Haghighi A, Zadeh NS, Sivard G, et al. Digital learning factories: Conceptualization, Review and Discussion[C]// Proceedings of the 6th Swedish Production Symposium

(SPS14). Gothenburg: SPS, 2014: 1-9.
[17] Ullrich A, Enke J, Teichmann M, et al. Audit-and then what? A roadmap for digitization of learning factories [J]. Procedia Manufacturing, 2019, 31: 162-168.
[18] Gronau N, Ullrich A, Teichmann M. Development of the industrial IoT competences in the areas of organization, process, and interaction based on the learning factory concept[J]. Procedia Manufacturing, 2017, 9: 254-2017.
[19] Grum M, Gronau N. Integration of augmented reality technologies in process modeling-the augmentation of real world scenarios with the KMDL[C]//Proceedings of the 7th International Symposium on Business Modeling and Software Design, BMSD. Barcelona: SciTePress, 2017: 206-215.

André Ullrich studied business administration at the University of Potsdam and the Finance Academy Moscow. He received his diploma degree in 2011 and his Ph.D. degree in Business Informatics from the University of Potsdam, Germany, in 2018. The emphasis of his work lies in sustainability of enterprise architectures, innovation processes, knowledge dynamics in digital environments and learning factories.

Malte Teichmann fulfilled his master of arts in educational science at University of Potsdam with a focus on vocational training and development of organizations. He develops learning modules and tasks for vocational training and examines the potentials of Tablets, Smartphones, and AR-Glasses for adult learning. Furthermore, he develops teaching/learning-scenarios in the Research and Application Center Industry 4.0.

Norbert Gronau studied engineering and business administration at Berlin University of Technology. He received his Ph.D. in Computer Science 1994 at Berlin University of Technology and finished his habilitation thesis there. Currently, he holds the Chair of Business Informatics, esp. Processes and Systems at the University of Potsdam, Germany. His main research activities concentrate on the areas of knowledge management and business process management. He is regular member of the German Academy of Technical Sciences, author of more than 90 papers and author resp. editor of several books.

(Publishing Editor: Yuan Zhao)