Classification of Empirical Programming Competencies in Introductory Programming Classes at German Universities according to the Anderson Krathwohl Taxonomy

Cognitive Processes	*Knowledge Dimensions*			
	Factual Knowledge	*Conceptual Knowledge*	*Procedural Knowledge*	*Metacognitive Knowledge*
Remember	• Assign literals to data types • Know elementary programming language constructs	• Know libraries for algorithms • Know basic characteristics of algorithms • Know terms and categories concerning complexity and efficiency of algorithms • Know elements of GUIs	• Know tools (IDEs, debugger, profiler) • Know the structure and principles of networks, the computer and other technological basics • Know how compiler and interpreter operate • Know distributed systems and parallel programming • know basic principles of programming languages • Know methods for the formal definition of programming languages • Know syntax and semantic of programming language • Know concepts for data management • Know mathematical basics of algorithms • Know basic algorithms and data structures • Know design pattern for algorithms and data structures • Know implementation methods for data structures • Process knowledge of run-time analysis • Know methods and tools for modeling of algorithms • Know methods of software development • Know quality criteria and conventions for source code	
Under-stand		• Describe concepts of programming paradigms • Explain problems that can be solved by algorithms • Explain terms of formal verification techniques	• Characterize algorithms, data structures and data types • Justify the use of software development tools and paradigms	
Apply			• Use computers • Document programs professionally • Apply quality criteria to source code • Perform mathematical calculations • Use existing libraries • Use tools for software development	
Analyze		• Characterize programming language and paradigms by analyzing their inner structure	• Break down given problems into smaller components • Analyze adequacy and characteristics of data structures • Analyze adequacy and characteristics of algorithms • Analyze algorithms and their complexity • Being able to read, explain and identify the output of (foreign) code • Comprehend compiler and interpreter messages	

Evaluate			• Assess adequacy of algorithms and data structures • Assess adequacy of solutions written in programming languages • Debugging of programs • Testing of algorithms and programs for errors • Evaluate properties of algorithms and programs • Assess the complexity of algorithms • Assess adequacy of programming tools and templates	• Self-reflection • Evaluate adequacy of one's own self-concept • Taking responsibility for learning processes
Create			• Design program specifications • Use programming language constructs correctly • Use logical expressions and operators to solve problems • Add code to a given method declaration • Write a single class • Write class(es) and a corresponding method • Write and call methods of an object • Generate objects • Write a program with several classes • Extend or adapt given program code • Write loops • Modeling of problems and programs • Design problem-adequate algorithms • Design smaller programs in a structured way • Design concurrently or parallel running processes • Design (software) projects in a structured way • Specify and implement (abstract) data types • Use and adapt standard data structures • Design data structures • Use and adapt standard algorithms • Implement algorithms • Write an executable program to solve a problem • Design interfaces • Programming of GUIs • Develop libraries	• Transfer of knowledge and skills to new problems • Self-regulated organization of learning process • Use of external resources for studying • Abstraction of rules (e.g. in recursion) and problems • Develop a systematic approach to problem solving