How Things Work – Recognizing and Describing Functionality

Michael Weigend
Institut für Didaktik der Mathematik und der Informatik
University of Münster
Fliednerstr. 21
48149 Münster, Germany
michael.weigend@uni-muenster.de

Abstract: Recognizing and defining functionality is a key competence adopted in all kinds of programming projects. This study investigates how far students without specific informatics training are able to identify and verbalize functions and parameters. It presents observations from classroom activities on functional modeling in high school chemistry lessons with altogether 154 students. Finally it discusses the potential of functional modelling to improve the comprehension of scientific content.

Keywords: Function, programming, parameter, competence, abstraction

1 Recognizing Functionality

In mathematics a function is a mapping, a relation between a set A of inputs and a set B of outputs. Each input from set A is related to exactly one output from B.

Computer scientists consider functions as a programming construct that is used to cope with complexity. A complex operation serving a certain purpose is divided in less complex operations, which are easier to implement by program text. This is a method of structural decomposition and a fundamental idea of computer science (Schwill, 1994).

All higher programming languages like Java or Python support defining functions. Technically a function definition consists of a function name, parameters and a block of instructions, defining in what way the parameters are processed in order to produce a result which (in many cases) is returned to the calling process in a special return statement.

There are functions that do not return any object explicitly (procedures) but change the state of a mutable object. For example a list may be extended by appending a new element. On a higher level of abstraction one might say that the old state of the mutable object is the input and the new state is the output. Functions may be stand-alone objects. In object-oriented programming they are connected to classes or instances of classes (class methods and methods). An object represents some holistic entity from real life or fantasy. The methods are related to the general meaning of the object. For example in Python the class list represents mutable sequences of items. The methods of list objects represent meaningful facets of the holistic concept of a list like inserting, removing, changing or appending items.

Let me now shortly discuss four properties of functions that are relevant for learners.

1.1 A function is different of structure

According to Kroes (1997) all technical artifacts have a structure and a function, which "has a meaning only in the context of intentional human action" (p. 291). The function of a clock is to tell the time. We need this to manage our lives. Its structure is its physical implementation by electronic components, power supply etc. The dichotomy of structure and function is also adopted in biology to describe natural systems. In physiology the human body is seen as an aggregate of organs, which have certain functions in relation to other organs. For example the function of the heart is to pump blood. A function in a computer program is not a physical but a digital artefact. Its dual nature is given a) by its purpose (the desired effect) and b) by its implementation consisting of a block of program statements. When a programmer decomposes a complex task into less complex subtasks by defining functions or a class structure, she or he focuses on functionality and ignores the implementation of the functions or classes. They are considered as black boxes.

1.2 Functions are abstractions

One and the same function can be used to describe different activities. Lakoff and Nunez (1997) discuss conceptual metaphors for arithmetic operations that are used in math education. For example the addition 4 + 3 can be represented by putting together a collection of four beads and a collection of three beads ("arithmetics is collecting objects"). Another metaphor for the same operation is walking four steps and then walking another three steps in the same direction ("arithmetics is walking along a line"). Creating a function is just the other way round. It is finding an abstraction of activities like putting together collections of beads and walking certain distances. (In this example it is inventing addition.) When a programmer creates a new function within a software project, she or he tries to create an abstraction that can be called several times. This strategy leads to an efficient development process.

Abstraction takes place when (existing) functions or operators are overloaded. For example, one can apply the concept of "adding" to different domains

- Numbers: 2 + 2 == 4
- Sequences : [1, 2] + [2, 3] == [1, 2, 3, 4]
- Colors: red + green = yellow.

Addition is an arithmetic operation, but in the context of sequences (strings, lists etc.) adding means concatenation. In physics, "adding colors" refers to mixing light of different colors (additive colors).

Technically, overloading means to reuse an already existing operator (like +) or function name (like len) for a new activity. In Python the name __add__ corresponds to the operator +. When you want to define an addition for objects of class C you define a method named __add__ within the class definition of C.

1.3 Functions represent holistic concepts

To be of help in a modelling process a function must represent a single holistic idea of activity. Most functions are labelled by one verb: to add, to append, to destroy. It is good style in computer programming to use meaningful names for all kind of objects. It is recommended that a function name should be a verb. Regarding the mental representation of a function a meaningful name is more than good style but essential. According to Baddeley (2003) humans can only handle a few chunks of information in working memory at the same time. A function call can be regarded as such a chunk. If the idea of a function is not fully understood and clear it must be rehearsed first before it can be used for problem solving. It is for instance impossible to create or to understand an algorithm based on adding numbers and extending lists, when the meaning of these operations is not perfectly clear. A reason for overloading an operator like + is that it represents a gestalt-like concept that is already familiar. It is easier to extend this to a new domain than to create something new.

1.4 Functions may have parameters

There exist functions without any parameters. Each constant object can be considered as a zero-ary function. But these are special cases. There are two prominent intuitions visualizing the idea of a function: factory and tool (see Weigend, 2007). The factory-model is a black box with an entrance for input data and an exit through which produced output data leave. The tool model visualizes the function as a tool (e.g. a knife) that is able to modify a mutable object (e.g. cut off something). However, the objects which are processed by the function are specified by parameters. A function call (like a metaphor) implies a transfer of knowledge from one domain to another. And parameters with meaningful names can support this cognitive operation. Parameters are used in a function call (as arguments) and represent objects from domain A, where the function is used. Corresponding parameters appear also in the definition of the function (formal parameters) and represent objects within the domain B of the function definition. Consider this simple function, which calculates the area of a rectangle (Python):

```
def area (length, width):
return length*width
```

The parameters represent objects from the domain *geometry*. Imagine to use this function for calculating the area of a rectangular door, which is appropriate for humans. The function call (with position arguments) may look like this:

```
area(height + 10, armspan)
```

The parameters are related to the physical properties of a human. Thus they are from a different domain: biology. The transition from one domain to another can be made more explicit by using keyword arguments (Python):

```
area(length=height+10, width=armspan)
```

Each keyword argument *key=value* includes a mapping from an item of domain A to an item domain B.

2 Modeling with Functions as a Competence

The four properties of functions discussed in the previous section, correspond to cognitive operations that a programmer has to perform in some way, when she or he creates functions or classes of objects in order to model a scenario.

- *Abstraction.* The programmer must find similar activities within the scenario, which can be modeled by the same function. This implies the ability to use only functional aspects (not structural) as criteria for classification.
- *Conceptualizing.* The programmer must find a concept that describes all activities of a category on a more abstract level. She or he has to find a meaningful name that labels the concept.
- *Parameterization.* The programmer needs to identify parameters, i.e. objects that are taken as input and processed by the function.

What kind of cognitive operations are performed, when a programmer uses functions that already exist in the repertoire of the programming language?

- *Deductive reasoning*. When a programmer browses through class libraries looking for an appropriate function or class she or he has to understand functionality described in the documentation and apply it to a new context.
- *Transfer of knowledge.* In programming literature functions belong to a context like a class or a library. For example, in Python 3.3 instances of the built-in class list have 33 methods (22 of them are overloaded operators and functions). All these functions are related to the concept of a linear sequence of objects, which could be pictured by – say a row of ten boxes. Imagine Jenny using a list to model a collection of airports. When she uses the function len() to calculate the number of airports, she transfers the term *length* from the image of a sequence of objects in a row (which has a certain length) to a new domain. Airports are not boxes laying in a row. The term *length* is now metaphorical.

According to Schwill (1994), fundamental ideas (like decomposition) can be explained and understood on a low level without specific computer science (CS) knowledge. This is an implication of the "vertical criterion". The major question this contribution is focused on is: How far are students without specific informatics training able to identify and verbalize functions and parameters related to objects in real life?

In the years 2013 and 2014 I have conducted a couple of classroom activities in a high school that were related to functionality. The students had to associate things from everyday life to laboratory equipment with the same functionality (abstraction), verbalize this common function (conceptualization) and name parameters (parametrization). For example, a glass tube has the same function as a trail, this function can be verbalized by the verb "to guide" and typical parameters are fluids in case of a tube and people in case of a trail.

Before I present more details of these classroom exercises let me briefly characterize the three facets of functional analysis from the perspective of Raymond Cattell's theory of fluid and crystalline intelligence (Cattell, 1963). Abstraction by classifying activities or tools, is related to fluid intelligence, since it does not require language skills. It is rather a general ability to solve problems in a novel situation independent from specific knowledge or experience. For example when Jenny associates a glass tube to a trail, she compares typical processes related to tubes and trails and finds a common principle. On the other hand, *verbalizing* functions and parameters implies a lot of crystallized intelligence, which is the ability to use skills, knowledge and experience. It is language-related and culture-dependent.

3 Activity 1: Functionality of Laboratory Equipment

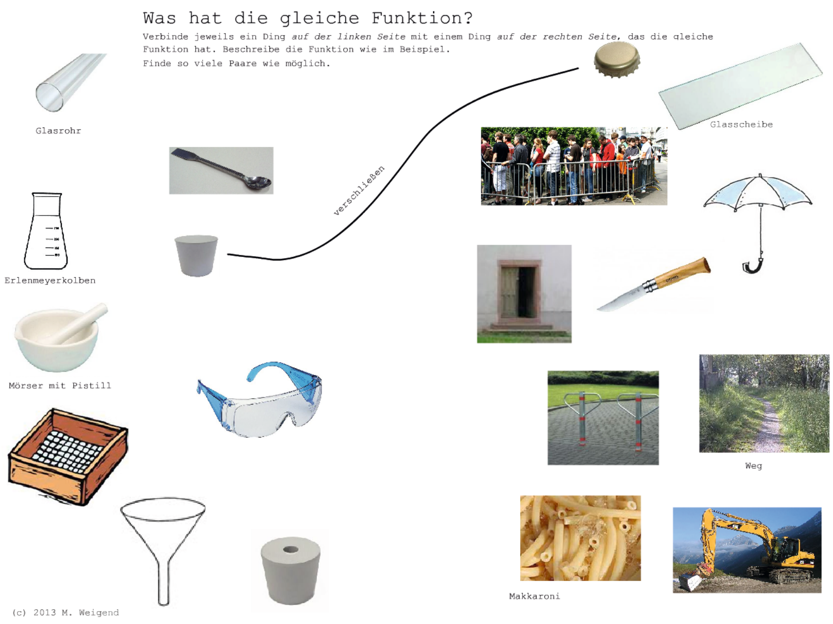
The participants got a worksheet depicting items from a chemistry lab (glass tube, spoon, Erlenmeyer flask etc.) on the left hand. On the right hand side there were things from everyday life. Although the items were from different domains and had different structures, some of them had similar functions. For example, a rubber plug and a crown cap look different and are made of different materials but they are both used to close containers to keep the content safe. The students' task was

1. to connect corresponding items by a line and
2. to name the function they have in common and write the words on the line.

The search for similar functionality corresponds to browsing through class libraries looking for appropriate functions for a software project.

75 high school students from grade 6 and 7 (age 11 to 14, average age 11.8, including 42 girls and 31 boys) were asked to perform this task. Beside the given example (rubber plug and crown cap with the function: to close) there were seven intended relations. Three images were meant as distractors and were not expected to be associated to anything from the complementary domain: Erlenmeyer flask, pasta, glass slide. The students found additional unexpected associations. For example, one person connected protective goggles with a knife and as a common function he called protection.

The students found an average of 6.5 pairs of corresponding objects and verbalized an average of 4.0 functions. Table 1 shows some results from the analysis of students' work.


Was hat die gleiche Funktion?

Verbinde jeweils ein Ding *auf der linken Seite* mit einem Ding *auf der rechten Seite*, das die gleiche Funktion hat. Beschreibe die Funktion wie im Beispiel. Finde so viele Paare wie möglich.

Figure 1: Worksheet from activity 1 “What has the same function?”

Table 1: Some results from “What has the same function?” (n=75)

Laboratory device	expected association (percent)	most selected unexpected (percent)	wording for expected functions (examples)	verbalizing a function, including unexpected	referring to structure (material, shape)
glass tube	trail (29%)	slide (25%), macaroni (25%)	guide, transport	44%	23%
spoon patel	excavator (65%)	knife (19%)	dig, pick, excavate	65%	0%
mortar	knife (69%)	excavator (9%)	destroy, crush	67%	0%
protective goggles	umbrella (65%)	glass slide (16%)	protect	85%	0%
sieve	barriers for cars (28%)	barriers at queue (28%)	sort out, prevent big things from entering	44%	5%
funnel	barriers at queue (16%)	barriers for cars (28%)	let through only a little	33%	10%
One-hole rubber stopper	door (22%)	macaroni (25%)	close, shut, bar	37%	16%

The second column (expected association) tells objects from everyday life that share a common function with a laboratory device from the first column and the percentages of students who have chosen this association. The third column shows the most popular unexpected associations. The students used a variety of phrases to describe the functionality. Column 4 tells a few examples.

In some cases students did associate objects because common structural features (like shape and material) instead of common functionality. For example some students connected a glass tube with macaroni and wrote "both have a hole in the middle". The last column shows the percentage of such misunderstandings.

The findings from this activity demonstrate that students in grade 6 and 7 are able to distinguish between structure and function, classify objects according to functionality and verbalize a function. Some functions (like the common function of a spoon and an excavator) are easier to identify than others (like the common function of a glass tube and a trail). Why are some connections easier to find than others? A possible reason could be the degree of abstraction involved. A simple approach to determine the level of abstraction would be to compare the parameters. The common function of a spoon and an excavator is to move portions of amorphous material (like sand or powder). The parameters are quite similar. The common function of a glass tube and a trail is to guide objects from one location to another. In this case the parameters are fluids resp. humans, which are very different. Torreano et al. (2005) define levels of abstraction for metaphors in a quite similar way by checking common elements in the metaphorical and the literal meaning of a phrase.

4 Activity 2: Functional Analysis of Electrolysis

57 students (31 boys, 23 girls, 3 did not tell the gender) from chemistry classes in grade 10 performed a functional analysis of an electrolysis apparatus consisting of a battery (power supply), ammeter, two electrodes in a U-tube with diaphragm, filled with a solution of copper chloride (see Fig. 2 in the middle).

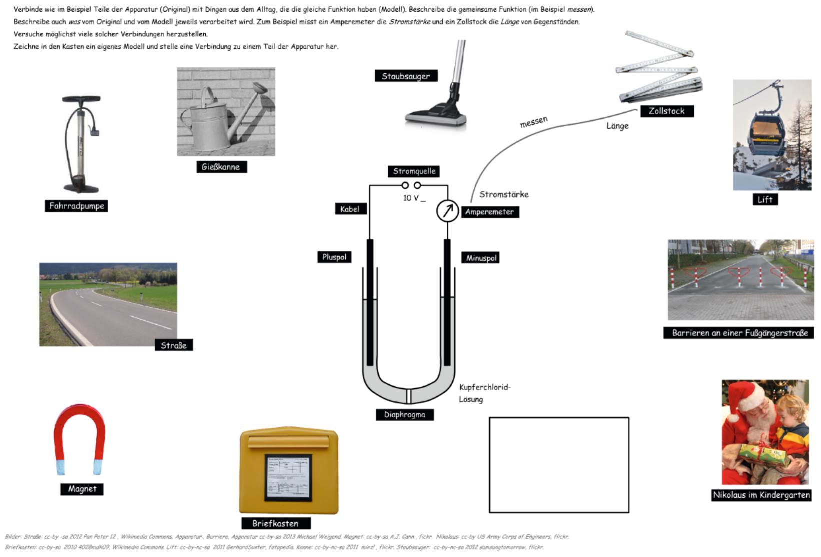
The students were asked to

- connect as many parts as possible from the electrolysis apparatus in the middle of the worksheet to objects from everyday life around it, which have the same function,
- verbalize the common functions,
- name the parameters at both ends of the connecting lines,

- draw an image of another object that shares a function with one of the part of the apparatus and connect it.

Before they started, the terms “function” and “parameter” were explained discussing the example given on the worksheet: Ammeter and ruler have the same function *to measure*. The parameter (the entity that is measured) is *electric current* in case of the ammeter and *length* in case of the ruler.

The worksheet suggested a functional decomposition of the electrolysis apparatus. A component may have several different functions. The negative electrode for example (1) attracts positive ions (like a magnet attracting iron) and (2) donates electrons to positive ions (like a person donating presents).


Verbinde wie im Beispiel Teile der Apparatur (Original) mit Dingen aus dem Alltag, die die gleiche Funktion haben (Modell). Beschreibe die gemeinsame Funktion (im Beispiel *messen*).
Beschreibe auch *was* vom Original und vom Modell jeweils verarbeitet wird. Zum Beispiel misst ein Amperemeter die *Stromstärke* und ein Zollstock die *Länge* von Gegenständen.
Versuche möglichst viele solcher Verbindungen herzustellen.
Zeichne in den Kasten ein eigenes Modell und stelle eine Verbindung zu einem Teil der Apparatur her.

Bilder: Straße: cc-by -sa 2012 Pan Peter 12 , Wikimedia Commons. Apparatur:, Barriere, Apparatur cc-by-sa 2013 Michael Weigend. Magnet: cc-by-sa A.J. Cann , fickr. Nikolaus: cc-by US Army Corps of Engineers, flickr. Briefkasten: cc-by-sa 2010 4028mdk09, Wikimedia Commons. Lift: cc-by-nc-sa 2011 GerhardSuster, fotopedia. Kanne: cc-by-nc-sa 2011 miez! , flickr. Staubsauger: cc-by-nc-sa 2012 samsungtomorrow, flickr.

Figure 2: Worksheet “Functional Analysis of Electrolysis”

Beside the example (ruler connected to ammeter) there were nine more images from everyday life to work on. On average the students drew 6.0 connecting lines and verbalized 4.1 functions. Only verbal expressions indicating purposeful activity (e.g. attraction, storage, to guide, to donate) were accepted as proper function names. Expressions referring to structural properties (e.g. to have a positive pole, to need electricity) were not. Whereas naming functions seems to be pretty much part of common knowledge at the age of 16, identifying parameters is not. The average number of parameter pairs was 0.8. Only 33 % of the students were able to name parameters at all. Parameters seem to mark a barrier, a transition from "common sense" to computational thinking that requires a special education.

13 students (23 %) created an additional image, 9 of them connected it to a part of the apparatus and 5 verbalized a function.

5 Activity 3: Functional Analysis of a Spectrophotometer

22 students from a high school chemistry class in grade 13 (5 boys, 17 girls, age 18 to 20) who had studied the principle of operation of a spectrophotometer were asked to perform a functional analysis of this device by connecting parts of the apparatus to objects from different domains with the same function (Fig. 3). Additionally they tried to verbalize the function and name parameters. At the beginning of this exercise the given example was explained: The common function of a tap and a light bulb is *to emit*. The parameters are *water* in case of the tap and *light* in case of the light bulb. The students discussed the matter in small groups and solved the task in a collaborative way.

There were seven parts and seven objects from everyday life to connect. Again, in some cases the students found not intended relations, which still might be considered to be reasonable.

However, the plausibility of the Table 3 shows some findings. Columns 2–4 tell how far the students were able to find a connection and name a common function and parameters, disregarding the plausibility (or correctness) of their choice. The last column tells the percentage of students who have chosen the intended pair of objects.

Later the students (n=21) evaluated the activity by rating the degree of agreement with some statements. They reported, that they talked about the spectrometer (average degree of agreement, ADA: 95 %), discussed at least one issue controversially (ADA 86 %), got a better understanding of a spectrophotometer (ADA 77 %) and had fun (ADA 76 %). Only a minority felt that it was not interesting (ADA 21 %) and took too much time (38 %).

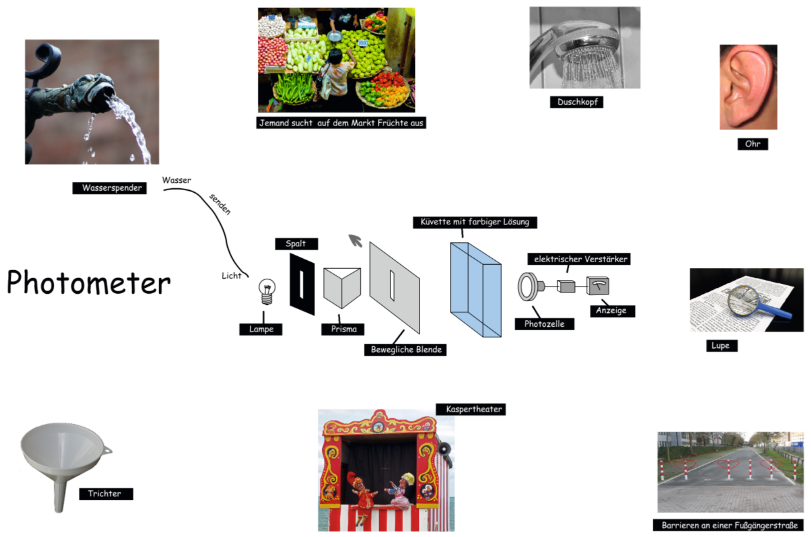


Figure 3: Worksheet „Functional Analysis of a Spectrophotometer".

Table 2: Some results from "Functional Analysis of a Spectrophotometer".

Part (corres-ponding object)	connection only	connection and function name	connection, name and parameters	having chosen the intended pair
Slit (funnel)	5%	14%	82%	59%
Prism (shower head)	0%	9%	91%	95%
Movable slit (person selecting fruits)	0%	23%	77%	50%
Cuvette with colored liquid (barriers)	14%	5%	73%	50%
photocell (ear)	0%	18%	82%	50%
Amplifier (magnifying glass)	0%	23%	77%	100%
Display (puppet theatre)	0%	23%	77%	50%

6 Benefits from Computational Thinking for Understanding Science

In the previous sections I have presented some empirical findings on students' competence of functional modelling. This competence is usually one of the major goals of computer science education at high schools. The idea of programming projects in the classroom is not to produce software specialist but to foster “computational thinking” (Wing) that is of use in wider areas of knowledge processing.

The question is: Does the ability of functional modelling help learning and understanding sciences like chemistry and physics? Andrea di Sessa (2002) illustrates the advantages of algebra for understanding physics. He presents Galileo’s original proofs of simple propositions in kinetics, which were written without any equations, since Galileo did not know algebra. These proofs are very difficult to understand. But every ninth-grader can proof the same propositions just by transforming equations, and gets some understanding this way. Students do not learn the competence of handling equations in physics but (basically) in math lessons.

What about functional modelling? Let me mention four issues:

1. Functional modelling by defining functions with parameters is a pattern that might help understanding structures. It is an approach to cope with complexity by decomposing that not limited to the design of software systems.
2. Science students sometimes mix up structure and function. In chemistry classes students sometimes say that a negative electrode attracts positive ions in a solution (like Cu2+) through “magnetic force”. In fact the physical cause for the movement of ions is “electric force”. But people have much more experience with magnets than with electrically charged bodies. Thus “magnetic attraction” is often just meant (metaphorically) as a *functional* concept (to attract = being magnetic). This can be clarified by a defining functions and parameters in a quasi-programming style. The pattern “a function processes parameters” forces to explicate the difference between a magnet and a negative electrode.
3. Misconceptions often remain in the dark, just because nobody talks about them. Collaborative functional modelling encourages discussions and explication of ideas. Here is an opportunity to make useful mistakes. This way misconceptions can be “diagnosed” and “cured”. This is comparable to using mathematical techniques to check the plausibility of a scientific calculations and chains of evidence.
4. The technique of functional modelling is a facet of computational thinking. It cannot just be noted like a piece of information, but it must be practised (rehearsed) and reflected, to be understood and to be of use. It is a competence. To develop this competence is an object of computer science rather than natural science education.

References

Baddeley, A. (2003): Working Memory Looking Back and Looking Forward. *Nature Reviews Neuroscience*, Vol. 4, 829–839.

Cattell, R. B. (1963): Theory of fluid and crystallized intelligence: A critical experiment. *Journal of educational psychology*, 54(1), 1.

Dehn, M. J. (2008): *Working Memory and Academic Learning*. John Wiley & Sons, Hoboken, New Jersey.

diSessa, A. A. (2001): *Changing minds, Computers, learning, and literacy*. MIT Press, Cambridge, MA.

Jaeggi, S. M., Buschkueh, M., Jonides, J., Perrig, W. J. (2008): Improving fluid intelligence with training on working memory. In *PNAS*, Vol. 105, No. 19, 6829–6833.

Kroes, P. (1998): Technological explanations: The relation between structure and function of technological object. In *Techné: Journal of the Society for Philosophy and Technology*, Vol. 3, No. 3.

Lakoff, G., Núnez, R. E. (1997): The Metaphorical Structure of Mathematics: Sketching Out Cognitive Foundations for a Mind-Based Mathematics. In Mathematical Reasoning. Analogies, Metaphors, and Images (ed. Lyn D. English). Lawrence Erlbaum Associates, Publishers, Mahwah, London, 21–92.

Schwill, A. (1994): Fundamental ideas of computer science. *Bulletin – European Association for Theoretical Computer Science*, 53, 274–274.

Torreano, L. A., Cacciari, C., Glucksberg, S. (2005): When Dogs Can Fly: Level of Abstraction as a Cue to Metaphorical Use of Verbs. In *Metaphor and symbol*, 20(4), 259–274.

Weigend, M. (2007): *Intuitive Modelle der Informatik [Intuitive Models of Computer Science]*, Universitätsverlag Potsdam.

Biography

Michael Weigend studied Computer Science, Chemistry and Education at the University of Bochum and at the University of Hagen and received a PhD in Computer Science from the University of Potsdam. He is a teacher at a secondary school in Witten, Germany and he has taught Didactics of CS at the University of Hagen for almost 20 years. He has published several books on computer programming, web development and visual modelling.