Editor: **Philippe Kruchten**
University of British Columbia
pbk@ece.ubc.ca

Thinking Fast and Slow in Software Engineering

Giancarlo Guizzardi, Oscar Pastor, and Veda C. Storey

THINK OF A child learning how to catch a ball repeatedly thrown to her by her father. As the child practices or continues with this activity, she becomes better at it. Through a process of trial and error and across several attempts, the child, in essence, is gathering more data on what works well and what does not work and, in this manner, mapping what she learns to the outcome (ball catching). If the child could possibly articulate what she learned, the result could be represented as a function: an extensional mathematical device mapping ball paths to the appropriate actions. With better and more and more trials, the accuracy of the function increases. Nonetheless, the function remains dependent on the available data (ball catching experiences).

©SHUTTERSTOCK.COM/TIERNEYMJ

Now, suppose that, after growing up, the child is able to understand what occurs when ball throwing, in terms of concepts and properties of relevant things in the world, e.g., gravity, the initial force of throwing the ball, throwing angles, air resistance, distance, and so on. That understanding could then be synthesized in one single equation (a symbolic artifact). Moreover, understanding (in terms of concepts) gives meaning to the elements in the equation, and it also explains why that equation (now an intentional artifact) can account for all the previous trials (data points) and all possible future trials. At that point, the data points themselves are no longer needed. The equation, as a symbolic intentional artifact, is all that one needs to predict how to behave in possible ball-throwing/catching circumstances. Note that the equation describing the possible movements of the ball describes this class of events but does not explain it. For the explanation, we need to refer to laws of nature and the concepts and properties populating the ontology of the domain.[1]

Data-Driven Versus Theory-Driven Knowledge

The ball-catching story illustrates the two complementary strategies that

Digital Object Identifier 10.1109/MS.2023.3306132
Date of current version: 1 December 2023

we, as humans, employ to acquire knowledge and reason about the world. We use a bottom-up approach starting from data (e.g., ball-throwing events) to generalize mathematical descriptions and, ultimately, conceptualizations that explain or identify patterns in the data. We also use a top-down approach, which starts by conceptualizing important concepts (e.g., physical agents having goals and physical objects, with their related mass, gravity, distance, and force). We then represent them so that appropriate data can be obtained that correspond to these concepts. In the top-down approach, we can test our conceptualizations against empirical data in the world, which, in turn, can trigger the evolution of our descriptions and conceptualizations.

These two strategies also bear some similarities to Daniel Kahneman's two systems of reasoning in his *Thinking Fast and Slow*. System 1 refers to forming (inexact, fallible, and perhaps biased) patterns from sense data (experiences). System 2 refers to sensemaking by connecting data to higher-level concepts via deliberative logical reasoning. As Kahneman writes, "The automatic operations of System 1 generate surprisingly complex patterns of ideas, but only the slower System 2 can construct thoughts in an orderly series of steps."[2]

Both data-driven (bottom-up) and theory-driven (top-down) forms of knowledge acquisition strategies are essential and can be captured in the following paraphrase of Immanuel Kant: "Concepts without data are empty; data without concepts are blind. Only in their unison can knowledge arise." However, from the perspective of an organism that wishes to reliably and efficiently transfer knowledge from one situation to the next, symbolic intentional artifacts (e.g., algorithms, systems of equations, and logical theories) that are explainable in terms of domain concepts are the highest form of knowledge representation structures our cognitive processes are able to produce. In contrast, vast amounts of data are an imperfect replacement when we are not able to synthesize knowledge in this form. In other words, although we need these two approaches working in tandem, and a data-driven strategy is essential for validating and evolving our theories of the world, the theories themselves are the most reliable and efficient form of knowledge we can produce.

Software Engineering

The software engineering discipline has historically loosely mimicked these two strategies. Classical software engineering uses a theory-driven (again, top-down, deliberate, and explicit) approach that starts by conceptualizing real-world concepts, relationships, properties, constraints, or stakeholders' goals. These lead to symbolic descriptions (i.e., programs) that ultimately lead to (runtime) data. Software tests, deployment, use, and maintenance then supply new data to correct and evolve both the underlying description and the conceptualization.

With the availability of big data and computing power, a new data-driven approach (bottom-up, inexact, possibly biased, and strongly based on subsymbolic artificial intelligence techniques) has become dominant. In this approach, there is no symbolic description of the data but, rather, a mathematical description of the regularities found in the data (basically, a complex extensional function) that is generated ("learned") from the data, most likely using machine learning techniques. The data-driven approach promises to generate software from data without the need to program, in other words, descriptions without a descriptor.

There is a fundamental asymmetry in the use of theory-driven and data-driven approaches in software engineering. This is in contrast to our theory-driven and data-driven cognitive processes, which do work in tandem. While in the theory-driven approach, the theory is always validated and corrected using real-time execution data, in our current data-driven practices, we never move from description (an executable program implementing a complex function) to conceptualization (an explanation of the program in terms of real-world concepts). As a result, in the latter approaches, we inevitably end up with software that is unexplained (and unexplainable). This is because explanation necessarily requires real-world semantics.[1,3] That is, it requires a mapping to concepts that are possessed by the explanation seeker.[1] Moreover, since software produced by a data-driven process is a function of the existing data (in which it was trained), it is not able to reflect on how its own instructions map to things and properties in the world, including human values and goals. Therefore, the software is unable to adapt to a large range of phenomena that can occur in reality. This step to conceptualization is missing, but necessary, for a software engineering practice to be able to make sense of data and deliver software systems that are trustworthy. However, software systems that are not trustworthy cannot be ethical.

Toward Meaningful Software

A software engineering strategy that relies solely on data, without considering connection to a conceptualization of the domain, is not guaranteed to produce software systems that are trustworthy and, hence, ethical.

In software engineering, we are increasingly required to produce software artifacts that impact human goals. For example, there are systems that decide whether mortgages are approved or parole granted. They can decide what information is disseminated and how and what actions are taken in cases of conflicts of interest or inconsistencies in the principles followed. Moreover, as discussed in Guizzardi et al.,[4] ethical requirements are always ecological requirements. That is, the stakeholders of these systems are not only the users directly interacting with the systems but the entire ecosystem of agents whose goals can be affected by the system. For example, autonomous vehicles do not involve only the passengers of a car itself. There are also other drivers, pedestrians, and citizens concerned with energy consumption as well as the cost of roads, unemployment, and so on. To be ethically designed, software systems must adhere to the principles of beneficence (contributing positively to the goals of these stakeholders), nonmaleficence (not diminishing the goals of these stakeholders), autonomy, explainability, and fairness.[4] Without explicitly considering these ethical requirements, we cannot produce autonomous systems that are truly explainable and exhibit beneficence/nonmaleficence.

Explainability is a key challenge to current data-driven approaches. This is expected: we have descriptions without a descriptor, so the meaning of these descriptions must be produced a posteriori. There are roughly two classes of strategies for explaining the behavior of these data-driven generating processes[1]: the white-box approaches and the black-box approaches. In the former, we aim at generating a symbolic artifact (typically, a decision tree) that mimics the behavior of that process. In the latter, we aim at explaining the process without opening the black box (e.g., via the description of counterfactual scenarios).

Now, consider beneficence/nonmaleficence. From the outset, it should be obvious that to preserve human and societal values, principles, and goals, especially under conflicting situations, software systems must be fully aware of what these values, principles, and goals are as well as how they influence one another. These essential components of ethical design will not emerge from the data themselves but must be explicitly elicited and modeled as a result of careful meticulous conceptual work.[4]

> To meet the challenge of producing ethical software, we need to consider the analogy with human cognitive systems and knowledge acquisition processes seriously.

In both cases, an explanation involves generating a symbolic artifact. However, as discussed at length in Guizzardi et al.,[1] these strategies rest on the false assumption that the symbolic artifacts generated are self-explanatory simply by virtue of being symbolic! Instead, and referring back to our initial story, we can understand, explain, and judge the quality (correctness and completeness) of our ball trajectory equations only by interpreting them in terms of domain concepts (forces, distances, throwing angles, and so on). There is no explanation without real-world semantics.[1,3] The challenging problem of semantics is exactly what is represented in the mapping from symbols to a conceptualization.

Furthermore, suppose it is possible to embed these ethical components into a software program, perhaps by explicit instructions or as learned rules. If an artifact operates in an open-world environment over which we cannot have control, then how can we guarantee that it will act with beneficence/nonmaleficence toward all relevant stakeholders under all possible conditions?

In short, we cannot. This is because of the nature of the descriptions that are generated in a bottom-up fashion. Specifically, the mathematical functions abstracted to account for specific datasets are extensional; i.e., their mappings reflect regularities present in those datasets. Again, referencing our story, if one generates a function merely based on one's

ABOUT THE AUTHORS

GIANCARLO GUIZZARDI is a full professor and chair of Semantics, Cybersecurity, and Services at the University of Twente, 7500 AE Enschede, The Netherlands. Contact him at g.guizzardi@utwente.nl.

OSCAR PASTOR is a full professor and Internationalization and Technology Transfer Director at the Valencian Research Institute for Artificial Intelligence, Universitat Politècnica de València, 46022 València, Spain. Contact him at opastor@dsic.upv.es.

VEDA C. STOREY is a full professor of Computer Information Systems at Georgia State University, Atlanta, GA 30303 USA. Contact her at vstorey@gsu.edu.

ball-catching experiences, that function cannot offer mappings from inputs under completely different conditions, for example, much heavier balls, ball throwing in very heavy wind, or ball throwing on Mars. To generalize beyond the actual history of experiences to experiences in new possible worlds, we need to contend with domain concepts and properties of the relevant things themselves. It is only by understanding these concepts and properties that we can map situations to behaviors beyond actual experiences and move from extensional descriptions to intentional descriptions. This is a key point for a simple reason: reality seems to present itself in a "long-tailed" fashion. In other words, most of our experiences exemplify only a small set of situation types. However, there are many situation types that can manifest themselves in different exceptional situations. If this is the case, then by simply gathering more data, we end up acquiring only more and more data about the same small set of situations.

In our current data-driven trend in software engineering, there are often hidden assumptions that all we need to produce software is heaps of data and general learning algorithms, that meaning will simply emerge from data, and that, by automating away human intervention, we can produce software at a faster pace. However, to meet the challenge of producing ethical software, we need to consider the analogy with human cognitive systems and knowledge acquisition processes seriously. Specifically, we require a proper integration of the software engineering equivalents of data-driven and theory-driven strategies. To have that, we must properly connect data to their symbolic descriptions with meaning, both in the sense of semantics and in the sense of purpose/significance. To achieve that, we cannot remove from software engineering the human-centric processes of domain conceptualization, conceptual analysis and clarification, semantic elaboration and ontological grounding, and ethical reflection, even if these processes seem to slow us down at first.

References

1. G. Guizzardi and N. Guarino, "Semantics, ontology and explanation," 2023, *arXiv:2304.11124*.
2. D. Kahneman, *Thinking, Fast and Slow*. New York, NY, USA: Farrar, Straus and Giroux, 2011.
3. K. Browne and B. Swift, "Semantics and explanation: Why counterfactual explanations produce adversarial examples in deep neural networks," 2020, *arXiv:2012.10076*.
4. R. Guizzardi et al., "An ontology-based approach to engineering ethicality requirements," *Softw. Syst. Model.*, early access, 2023, doi: 10.1007/s10270-023-01115-3.