Scientific Research Publishing

Journal of Software Engineering and Applications, 2017, 10, 168-173
http://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

How Do You Know What You Know: Epistemology in Software Engineering

Oluwatosin Ogundare

Industrial and Systems Engineering, Texas Tech University, Lubbock, TX, USA
Email: tosinogundare@hotmail.com

How to cite this paper: Ogundare, O. (2017) How Do You Know What You Know: Epistemology in Software Engineering. *Journal of Software Engineering and Applications*, **10**, 168-173.
https://doi.org/10.4236/jsea.2017.102010

Received: January 20, 2017
Accepted: February 21, 2017
Published: February 24, 2017

Abstract

Ubiquitous computing emphasizes the notion of automation in the daily human experience. With the ease, comes the responsibility of knowing, the knowledge of the intrinsic nature of the machine and the evolution of Human-Computer Interaction (HCI). The quest for knowledge is inevitable and exists even in the mundane experience. This innate search is the soul of the act of questioning. "How?" "What? "Why?" dominate our communal vocabulary and model the endlessness of our natural inquisitiveness. For example, an interaction of software systems in the case of a user who withdraws money from the ATM and automatically gets a text message and an e-mail containing notification of the transaction, engenders questions about how it all works; *i.e.*, the nature of the special science that enables wireless communications. The focus is establishing foundational Truths for the modern human-software co-existence. The discussion that follows involves a delineation between the function that an intelligent software system performs and the knowledge it implicitly encapsulates. The paper expounds the role of ontology in formalizing knowledge in software systems and its contribution to the unveiling of the mystical black box that intelligent software systems often present to their human counterpart.

Keywords

Software Engineering, Epistemology, Ontology

1. Introduction

"Knowledge" according to Plato is "Justified True Belief". Knowledge that derives from aphorisms or other self-evident truths is easier to acknowledge. For example, the knowledge of "multiplication" is justified by the truthfulness of "addition"—which is often referred to as the Apriori. However, in Software Engineering, the Apriori is more obscure. The investigation of the nature of knowledge in Software Engineering requires an expansion of the general idea of the

Apriori in establishing knowledge.

First, we examine the two types of technical knowledge that exist *viz.* Engineering Knowledge and Scientific Knowledge. The delineation of engineering knowledge to emphasize its distinction from scientific knowledge forms the premise of its understanding and highlights its own importance. According to Antonio Dias de Figueiredo, engineering knowledge is multidimensional and every notion it portrays can only be completely understood when analyzed as such [1]. So how is engineering knowledge constructed and what is its nature? Diana Forsythe "an anthropologist studying a scientific community engaged in formulating knowledge descriptions" asserts that knowledge in the applied sciences (engineering) is established through acquisition and formalization [2]. "Knowledge Acquisition" according to R. Studer *et al.* can be realized in one of two ways:

1) Transfer Process

2) Modeling Process

The Transfer and Modeling processes demonstrate some of the differences in constructing scientific and engineering knowledge as discussed subsequently.

2. Examining the Differences between Scientific & Engineering Paradigms

There is a marked distinction between knowledge realized through scientific methods and knowledge established from an engineering process. The question is how are they different? The difference is in the approach. Scientific theories or paradigms are largely based on controlled empirical processes while engineering methods are derived predominantly from the experience of what works. **Figure 1** suggests that while both paradigms begin as a search for the solution to a problem, the steps taken to find answers are very different [3].

The 10 steps in the engineering paradigm will in fact be an example of knowledge acquired through the "modeling" process. This process is widely used and accepted in modern enterprise software engineering. As software gets more focused on problem solving for mundane or everyday tasks, the need for a formal characterization of its encapsulated knowledge models, workflows and limitations is increasingly more important.

The Scientific Paradigm	The Engineering Paradigm
1. Problem Definition	1. Problem Definition
2. Identify Variables	2. Fact Finding
3. Formulate Hypothesis	3. Establish Parameters
4. Define Methodology	4. Define Acceptance Criteria(s)
5. Design Experiment	5. Define System Boundaries
6. Perform Experiment	6. Prototype Design
7. Test Hypothesis	7. Build Prototype
8. Analyze Results	8. Test
9. Draw Conclusions	9. Verify
10. Present Results	10. Present Results

Figure 1. Differences between scientific and engineering knowledge [7].

3. Ontologies in Software Engineering

"*Software engineering is one of the most knowledge-intensive professions. Knowledge and its management are relevant to several aspects of software engineering at different levels, from the strategic or organizational to the technical*" [4]

If engineering is largely viewed or broadly considered to be the application of pure "hard" science(s), then it follows that "Software Engineering" can be simply viewed as the application of principles or theories from the discipline of computer science to real world problems. However, IEEE gives the formal definition of "Software Engineering" as follows:

"… *It is the application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software*" [5]

From the definition, the adjectives "systematic" and "quantifiable" implies there must be a set of agreed upon rules by which measurement is carried out. Cuadrado-Gallego *et al.* suggest that the development of such global quantifiers will be impossible in Software Engineering without ontology. In fact, he says that without these formal ontologies, there will be no "shared, consensual conceptualization" in Software Engineering [6].

There are different types of Ontologies in Software Engineering, each of them serving different purposes. For example, Reference ontologies, whose main purpose is to eliminate ambiguities in terminology and mitigate the occurrence of what Thomas Kuhn refers to as "local incommensurability" [7]. The role of philosophy in creating such standard terminologies is elegantly stated by Nicola Guarino "… *philosophy and linguistics play a fundamental role in analyzing the structure of a given reality at a high level of generality and in formulating a clear and rigorous vocabulary.*" [8]

In contrast however, Application Ontologies are not as dense as its counterpart and thus do not require the detail and structure that would involve either Philosophy or Linguistics (at least in a formal sense). Application ontologies can be generally viewed as taxonomies of domain specific vocabulary. Apart from establishing commensurability, ontologies are relevant to epistemology in Software Engineering because they help establish Fidelity. This is especially important because according to Zelkowitz and Wallace, at least 30% of papers published on Software Engineering are not based on empirical science. Empirical methods are how we establish justified Truths or Knowledge in many science, applied science and quantitative fields where mathematical proofs are less applicable. The accepted pathway to science begins with a reasonable hypothesis, that is then tested rigorously without bias and its conclusions accepted as Knowledge. In modern software development, however, an intuition or a cross-pollinated idea from another discipline is implemented as an algorithm without ever having established its own truthfulness. Forsythe expresses a similar sentiment about her interaction with AI engineers in the excerpt below:

"*Talking with AI professionals, I had been struck by the apparent parallels*

between the process that they call 'knowledge acquisition' … and what anthropologist do in the course of field research … Asked how they went about the task of gathering knowledge for their expert systems, the knowledge engineers I met tended to look surprised and say, 'We just do it' …". [2]

Furthermore, a distinction should be made between what is considered a reasonable output of an intelligent system and what is verified knowledge produced by the same system. This is only possible if there is a formal knowledge acquisition and verification system integrated into the design of intelligent systems. In this regard, developing ontologies tie into the quest for establishing the foundation of knowledge in intelligent software systems. Ontologies in Software Engineering formalize the domain taxonomies, which leads to asking the right questions, establishing Truth and derived knowledge in line with what philosophers (empiricists) have always maintained. Furthermore, ontologies help establish technical aphorisms which may constitute Apriori truths in Software Engineering.

3.1. Reference Ontologies

Reference Ontologies (ROs) focuses on presenting generalized expressions (uses quantified variables in statements in line with First-Order logic) in favor of orthodox propositions (propositional logic) and is grounded in both metaphysical and epistemological realism. The central theme of Reference Ontologies is establishing Truth. However, Reference Ontologies can be wrong [9].

3.2. Application Ontologies

Application Ontologies focus on reasoning and is localized to a specific computational application. Application Ontologies are philosophically grounded in pragmatism. In essence, it admits knowledge of a solution to be whatever works in the resolution of the problem. Application Ontologies have a strong methodological emphasis on fidelity *i.e.* maintaining consistent expressions within a specific computational application. However, Application Ontologies also admit as equivalent, pragmatic alternatives to metaphysical realism, to enable their adoption in real world computational applications [9]. For example, a mathematical integration problem in calculus with limits in the closed range $[-\infty, +\infty]$ cannot be computationally realized but an approximate solution using a numerical method is admitted by an Application Ontology as equivalent. This form of informal equivalence relation might be problematic in mission critical intelligent systems with very limited error tolerance. In the case of computing the solution of an indefinite integral, approximations are inevitable for real world applications but the weakness of Application Ontologies in establishing knowledge is its tendency to admit approximations even for realizable computational problems. A reasonable conclusion from the preceding arguments is that the foundation of establishing knowledge in engineering and specifically Software Engineering begins with the definition of a formal Ontology. In the case of Software Engineering, we may prefer to define a Reference Ontology.

4. "Knowing" in Software Engineering

Following the discussion on the foundational notions of establishing knowledge and the formalization of ontologies as an important step in this regard, it is important to understand what can be potentially qualified as "known" and how they can be constructed and validated in Software Engineering.

4.1. The Nature of What Can Be Known in Software Engineering

Knowledge in Software Engineering is primarily acquired through modeling. The directed abstraction of "real world" problem solving without the interference of computational or software constructs or artifacts [10]. Constructing knowledge from modeling, according to Guus, is done at a conceptual level. This abstraction is grounded in the metaphysical nature of a Reference Ontology (RO), the goal here is not applicability but foundational Truth. Along these lines, the model must maintain internal consistency [11].

4.2. Knowledge Models in Software Engineering

Knowledge models in Software Engineering can be realized as a group of closed-form mathematical transformations composing an algorithm mimicking a problem solving "use case" in a specific problem domain. Otherwise, it could also be viewed as a logical combination of rules, like the "if-else-then" method (popular in expert systems), with each branch yielding an Apriori Truth or eventually leading to one. Regardless of how the model is realized, it is important that it is grounded in ontology and maintains internal consistency.

4.3. Validating Knowledge Models in Software Engineering (SE)

Establishing knowledge requires validation. Validating knowledge models can be viewed as providing justification for the model in reality. According to Kathleen Carly, validating models (computational models, engineering models, etc.) can be assessed at different levels targeting distinct aspects of the model. She identified at least eight aspects of the model that requires validation [12]. A summary of the prescribed knowledge validation techniques is presented as follows:

Face-Value validation: This measures how closely the model resembles the actual problem solving method in the real world [12].

Parameter Validation: This measures how closely the model parameters fit real world parameters [12].

Process Validation: This measures how closely the process described by the model resembles the actual real world process [12].

Pattern Validation: This measures if the pattern of outputs generated by the model exhibit the same pattern as in the real world [12].

Theoretical Validity: This checks the model for adherence to established theoretical constructs and procedures [12].

5. Conclusion

Guus Schreiber claims that attempts at describing the nature of knowledge are

irrelevant to its application to "problem solving" or inventing new technologies. His justification lies in the perceived inarticulate nature of engineers when asked to expound a topic they have learned, compared to scientists, fall short in providing rigorous detail yet constantly apply these concepts to building faster and more efficient technologies. Essentially, he seems to be saying that we do not need to be able to explain knowledge to apply it [10]. In spite of the truth of the aforementioned, without the ability to learn the nature of knowledge, construct and validate it, establishing fidelity in AI and Software Engineering models will remain problematic. Sound epistemology will open up the future to even more engineering marvels.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Figueiredo, A. (2008) Toward an Epistemology of Engineering. In: 2008 *Workshop on Philosophy and Engineering*, The Royal Academy of Engineering, London, 94-95.

[2] Forsythe, D.E. (1993) Engineering Knowledge: The Construction of Knowledge in Artificial Intelligence. *Social Studies of Science*, **23**, 445–477. https://doi.org/10.1177/0306312793023003002.

[3] http://research.uc.edu/sciencefair/resources-forms/topic-suggestions/scientific-method-v-engineering-design-procedures.aspx

[4] Aurum, A., *et al.*, Eds. (2013) Managing Software Engineering Knowledge. Springer Science & Business Media, Berlin.

[5] IEEE (1990) IEEE Standard Glossary of Software Engineering Terminology. IEEE Std.610, 12-1990.

[6] Cuadrado-Gallego, J., Rodríguez, D., Garre, M. and Rejas, R. (2007) Epistemological and Ontological Representation in Software Engineering. *International Conference on Computational Science*, Springer Berlin Heidelberg, 1162-1169. https://doi.org/10.1007/978-3-540-72586-2_162

[7] Kuhn, T.S. (2012) The Structure of Scientific Revolutions. University of Chicago Press, Chicago. https://doi.org/10.7208/chicago/9780226458144.001.0001

[8] Guarino, N. (1998) Formal Ontology and Information Systems. *Proceedings of FOIS*, **98**, No. 1998.

[9] Menzel, C. (2003) Reference Ontologies-Application Ontologies: Either/Or or Both/And? *KI Workshop on Reference Ontologies and Application Ontologies*, *CEUR Workshop Proceedings*, Vol. 94, Hamburg,

[10] Schreiber, G. (2000) Knowledge Engineering and Management: The Common KADS Methodology. MIT Press, Cambridge.

[11] Wielinga, B.J., Schreiber, A.Th. and Breuker, J.A. (1992) KADS: A Modelling Approach to Knowledge Engineering. *Knowledge Acquisition*, **4**, 5-53. https://doi.org/10.1016/1042-8143(92)90013-Q

[12] Carley, K.M. (1996) Validating Computational Models. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.195.6257