# Linking Software Engineering concepts to upper ontologies

Miguel-Angel Sicilia, Juan J. Cuadrado-Gallego

University of Alcalá, Ctra. Barcelona, km.33.6, 28871
Alcalá de Henares (Madrid), SPAIN
msicilia@uah.es, jjcg@uah.es

**Abstract.** Recent relevant efforts to shape the knowledge body of the discipline of Software Engineering have resulted in the crafting of different conceptualizations and ontologies both for specific or general purposes. But the underlying semantics of such knowledge representations are not defined formally, which would eventually require an effort of "harmonization" or mapping of knowledge structures, and results in non-explicit computational semantics. This can be avoided or alleviated by the reuse of existing open upper ontologies. This paper sketches how some of the fundamental Software Engineering terms can be mapped to the large *OpenCyc* knowledge base. The tentative mapping described provides a sound basis for analysis and discussion between different alternatives to the creation of Software Engineering conceptualizations.

## 1 Introduction

The growing interest in ontology-based applications as opposed to systems based on information models – in the sense given by Welty and Guarino (2001) – have resulted in a renewed interest in the definition of conceptual models for any kind of domain. *Software Engineering* is one of the domains that have received some previous attention in that direction (Mendes and Abran, 2004; Wille et al., 2003; Sicilia, Cuadrado and Rodríguez, 2005). The SWEBOK guide provides a foundation for the development of an ontology for Software Engineering, since it is the result of a process of domain expert review and validation, and provides references to other relevant sources. Nonetheless, the process of analysis of the guide to come up with a logical coherent ontology is by no means a simple process. Many of the entities described in the guide to the SWEBOK are complex activities that produce interrelated artifacts. These entities have temporal, material and conceptual facets that should be clearly defined, and which are well-known in existing upper ontologies and large commonsense bases. Furthermore, there exist proposals for the standardization of upper ontologies (Niles and Pease, 2001). In fact, the IEEE P1600.1 Standard Upper Ontology Working Group (SUO WG[1]) is working towards that end. Given the past activity of the IEEE and other organizations in producing standards regarding the

[1] http://suo.ieee.org/

vocabulary and concepts of Software Engineering, there exists an opportunity to exercise and analyze the discipline from the perspective of upper ontology as a principal case study.
A technique for validating the semantic precision of conceptual schemas is that of providing explicit links to concepts and relations that are already described in a large upper ontology. Concretely, we here consider the *OpenCyc* 0.9 knowledge base. This is an alternative to analysis techniques as the Bunge-Wand-Weber (Wand and Weber, 1995) that fosters the reuse of existing open knowledge engineering, and the mapping to modern Web-enabled ontology languages as OWL is a straightforward step.

*OpenCyc* is the open source version of the *Cyc* Knowledge Base (Lenat, 1994), which contains over one hundred thousands atomic terms, and is provided with an associated efficient inference engine. *Cyc* uses as its underlying definition language a variant of predicate calculus called *CycL*, and it attempts to provide a comprehensive upper ontology of “commonsense" knowledge.

In the rest of this paper, some of the principal concepts that surround the Software Engineering discipline are linked to *OpenCyc* definitions, as an illustration of the approach.

## 2. Mapping of core concepts to *OpenCyc*

Software engineering as any engineering discipline is mainly concerned with *how* things should be done. Thus, a large amount of SE theory is related to **activities** carried out by agents (the engineers). Further, every activity uses or creates **artifacts** of a various kind. A prominent category of such artifacts are **models**, which pervade current practice of SE. In what follows, an analysis of the representation of the three words in bold face above with regards to *OpenCyc* concepts will be sketched (ontology concepts are provided in `Courier` font).

### 2.1. Activities and activity prescriptions

Software engineering activities as actually enacted should be represented as dynamic (i.e. temporal) situations, represented by the term `Event`. Events in *OpenCyc* have a temporal extent but also “parts” of any kind (participants, places) that are modeled as predicates called `Role`s. Roles in turn can be further specified by *constraints* and relations between roles. Further, there are two important role categories: *actor* slots and *sub-event* slots, which respectively model “things involved” and the hierarchy of events that comprise a higher level one.

Regarding actor slots, they are binary predicates that connect an `Event` to `SomethingExisting`. *OpenCyc* provides a large number of built-in predicates that can be reused.

A ontologically different concept related to activities in SE is that of “methods” for activities, i.e. the normative specification of “blueprints” for potential courses of activity. These specifications have an intrinsic prescriptive character, so that they should not be specified as actions, but rather as specifications.

### 2.2. Artifacts

`Artifacts` are "at least partially tangible thing which was intentionally created by an Agent to serve some purpose or perform some function". *SoftwareEngineering* artifacts require further restriction, e.g. created by engineers, as part of an engineering activity and the like. Some additional restrictions are required; the following is a rough sketch:

- `ComputerDataArtifacts` may represent backups or prepared initialization data as produced by engineering.
- `ComputerProgramModule-CW` can be assimilated to the notion of *component* in a broad sense, including routines. Nonetheless, further specification of kinds of components is required.
- `ComputerCodeSource` and `ComputerCodeBinary` are self-evident, and there are other categories to represent interpreted code. The relation to the conceptual work is reified through `ComputerProgram-CW`. This category of artifacts has a rather comprehensive representation in *OpenCyc*. There are also predicates to relate source to binaries.

### 2.3. Models

The word model amounts for 297 occurrences in the SWEBOK guide. `Model-Artifact` provides the appropriate semantics for the concept: "a collection of artifacts; a subset of `VisualInformationBearingThing`. Each element of Model-Artifact is a tangible object designed to resemble and/or represent some other object, which may or may not exist tangibly". The `ModelFn` function designates all the models of a given thing, e.g. `ModelFn(SoftwareComponent)`. This is a concrete characterization of models that seems to match all the uses of model in the SWEBOK. As information bearing objects, the models are IBTs also, so that their contents can be represented in a propositional form, through the predicate `containsInfoPropositional-IBT` IBT PIT, that links to a propositional information thing. PITs are in themselves microtheories, thus allowing the definition in logical terms of the actual contents of the model. This could for example be applied to develop systems that represent UML diagrams through logics, which will enable a degree of increased automation.

The Guide to the SWEBOK somewhat differentiates models and artifacts, as in the Software Design KA "The output of this process is a set of **models and artifacts** that record the major decisions that have been taken", but ontologically this distinction is irrelevant.

## 3. Outlook

The mapping of some essential Software Engineering concepts with definitions in the large *OpenCyc* knowledge base has been described. This provides a sound foundation for the engineering of a comprehensive formal ontology of the discipline, integrating existing fragmentary efforts (Mendes and Abran, 2004; Wille et al., 2003;

Sicilia, Cuadrado and Rodríguez, 2005). The main challenge ahead is that of coordinating a process of systematic development of a domain ontology for Software Engineering based on guides as the SWEBOK that provides an epistemologically adequate representation of current practice and theory.

## References


Lenat, D. Cyc: A Large-Scale Investment in Knowledge Infrastructure, *Communications of the ACM* 38(11) (1995) 33-38.

Mendes, O., Abran, A. (2004). Software Engineering Ontology: A Development Methodology. *Metrics News*, 9, 68-76.

Niles, I., and Pease, A. 2001. Towards a Standard Upper Ontology. In Proceedings of the *2nd International Conference on Formal Ontology in Information Systems* (FOIS-2001), Chris Welty and Barry Smith, eds, Ogunquit, Maine, October 17-19, 2001.

Sicilia, M.A., Cuadrado, J.J., Rodríguez, D. (2005). Ontologies of Software Artifacts and Activities: Resource Annotation and Application to Learning Technologies. In *Proceedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering*

Wand, Y.; Weber, R. (1995) On the deep structure of information systems. *Information Systems Journal* (5), 1995, pp. 203-223.

Welty, C. and Guarino, N. Supporting ontological analysis of taxonomic relationships. *Data and Knowledge Engineering* 39(1), 2001, pp. 51-74.

Wille, C., Abran, A., Desharnais, J.M., Dumke, R.R. (2003). The quality concepts and subconcepts in SWEBOK: An ontology challenge, in *Proceedings of the 2003 International Workshop on Software Measurement (IWSM)*, 113--130.