Journal of Software Engineering and Applications, 2014, 7, 530-539
Published Online May 2014 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2014.76049

Cognitive Software Engineering: A Research Framework and Roadmap

Zohair Chentouf

College of Computer and Information Science, King Saud University, Riyadh, KSA
Email: zchentouf@ksu.edu.sa

Received 17 April 2014; revised 10 May 2014; accepted 17 May 2014

Abstract

The work of software engineers is inherently cognitive. Integral to their duties is understanding and developing several artifacts. Each one is based on a specific model and a given level of abstraction. What distinguishes Software Engineering is the logical complexity of some artifacts (especially programs), the high dependency among them, and the fact that the success of the software project also depends on the human and social factors, which characterize the engineers as individuals and as a group. The complexity of the daily tasks within a software development team motivates the investigation on the relevance of automating the software professionals' cognitive processes in order to make their work easier and more efficient. The success of this endeavor is expected to emerge as Cognitive Software Engineering. For this aim, the present article suggests a research framework and roadmap, which build on the current state of the art. Some future directions in the Cognitive Software Engineering are presented.

Keywords

Software Engineering, Cognitive Systems, CASE Tools, Cognitive Software Engineering

1. Introduction

Software projects are known to run behind schedule. This is due to the complexity of the daily tasks assigned to the software engineers. The complexity emanates from many aspects. For example, the natural language, which is the main communication means, often introduces ambiguity in communication and requirements. The complexity of programs' logic leads to defects and mistakes. The several design models which usually represent the same entity from different perspectives and at different abstraction levels make it difficult to understand the whole system structure and the flow of control and data between the different components of the software to build. The high number of risks, requirements, constraints, goals, design options, and implementation alternatives make it difficult to decide about priorities, mitigation plans, and technology and implementation options. The variable performance, mood, expectations, and social skills of the stakeholders and software team members create other problems that often lead to the poor productivity too.

All these difficulties are dealt with by the software team as cognitive agents. In this context, a two-fold research objective raises itself: reduce the cognitive burden for software engineers and make their cognitive tasks more efficient. Task automation is an obvious means for this aim. Task automation in Software Engineering exists and covers many activities of the software development process. The set of tools that are used are called Computer Aided Software Engineering (CASE) tools. However, as we will explain in Section 3, most of the current Software Engineering task automation does not apply on the human cognitive processes that software engineers employ in their daily duties. The objective of the present paper is to suggest a research framework and roadmap to bring more maturity to CASE tools in order for them to be cognitive, and for Software Engineering to become *Cognitive Software Engineering*. This leveraging principle of the present investigation marks out its research plan, which is centered around analyzing at which extent the current CASE tools fulfill the requirement of embodying the cognitive tasks of software engineers. As result of this analysis, the present article identifies the daily tasks and skills of software professionals, elaborates a *maturity model* of CASE tools, and suggests a metric to measure the *cognitive gap* that separates CASE tools and Software Engineering as a discipline from the objective of being cognitive.

Section 2 lists the Software Engineering tasks and skills. Section 3 presents a cognitive maturity model of CASE tools and elaborates a method to measure their maturity degree. Section 4 presents a research plan for Cognitive Software Engineering and analyzes its epistemological status. Section 5 concludes the article.

2. Software Engineering Tasks and Competencies

In order to be able to analyze CASE tools, we first need to establish a list of the Software Engineering daily tasks and competencies. For this aim, we considered two data sources. First, a set of emails and documents which have been used by the author of the present article during more than twelve years of software industry experience as developer, tester, architect, designer, team leader, project manager, and R & D director, have been analyzed in order to derive the set of Software Engineering tasks and competencies. The completeness of the derived list was then validated using the competency list of the Career Space. The latter is a European Union initiative, which gathered a consortium of eleven major IT and Telecommunication companies like BT, Cisco, IBM, Microsoft, Nokia, and Siemens. This consortium's aim was to address the problem of IT professionals' shortage in Europe. Among their objectives was the elaboration of a list of competencies. For that, the consortium first determined a set of professional job profiles and then determined the required set of competencies for every profile. The competencies of the Career Space are listed in [1]. Our list of competencies is presented in **Table 1**. As the reader may notice, the list does not contain general skills of project management or quality assurance. That is because the original list of the Career Space contains project management and quality assurance skills that are specific to IT and Software Engineering. This perfectly fits our objective in the present research. Another characteristic of the list of **Table 1** is that the competencies are at a level of abstraction that is lower than the Career Space's abstraction level. We chose this level of detail in order to be able to thoroughly analyze CASE tools based on these competencies.

3. Maturity of CASE Tools and Software Engineering

The essence of the software professionals' cognitive work can be modeled as *input-processing-output*. Inputs are data or more complex artifacts. The engineer may need to cognitively process data or artifacts in order to prepare the suitable representation of inputs. We consider such a task as part of processing. For example, in order to detect inconsistencies in a set of requirements, the analyst may need to re-write the requirements in a given syntax before reasoning about inconsistencies. We consider rewriting and reasoning as a single cognitive task. We do not consider the manual entry of data of any kind as cognitive. Outputs can be data or artifacts, not processing of any kind, with variable complexity degrees. CASE tools are expected to either assist or autonomously perform processing. Based on this model, we identify four complexity levels of CASE tool operation, which we consider as maturity levels (*ML*) as well:

- *ML*-0 (non-automated): the engineer performs processing.
- *ML*-1 (assisted): the tool performs part of processing under control and monitoring of the engineer.

Table 1. Software engineering competencies.

Phase	Competency	*ML*	Reference
Requirements	Eliciting requirements	0	
	Specifying requirements	2	Acclaro DFSS
	Detecting requirement inconsistencies and incompleteness	2	Requirement Composer
	Solving requirement inconsistencies and incompleteness	0	
	Identifying requirement rationale	1	Cognition Cockpit
	Identifying requirement dependencies	1	Blueprint
	Source traceability	0	
Design	Graphically modeling systems, sub-systems, components, interactions, control, and data flow	1	IBM-Rational
	Writing use cases	1	Acceleo
	Identifying dependencies with requirements	1	Cameo Req+
	Identifying dependencies with use cases	1	IBM-Rational
	Designing data models	1	ARCAD
Programming	Coding in conformance with the planned structure	1	IBM-Rational
	Unit testing	1	Eclipse
	Compiling	3	Eclipse
	Debugging	1	WDK
	Repairing defects	1	Argo/UML
	Program comprehension	1	Imagix 4D
	Code optimization	1	Via/Renaisance
	Peer code reviewing	1	Via/Renaisance
Testing	Designing tests/ installation/ integration	1	Via/Smarttest
	Identifying dependencies between test cases, requirements, and design	1	IBM-Rational
	Planning and scheduling test types based on current constraints	0	
	Configuring and scheduling automatic tests	2	Empirix
	Executing tests/ installation/ integration	1	AnthillPro
	Measuring and interpreting test results	1	AgileLoad
	Classifying defects	0	
	Checking documentation completeness and conformance	0	
Managing	Evaluating change requests	0	
	Executing client support	1	Mantis
	Applying version control management	1	CVS
	Choosing suitable technologies	0	
	Planning the software development cycle	1	AceProject
	Prioritizing requirements	0	

Continued

	Creating deployment and maintenance plans	1	CollabNet
	Specifying market requirements or enterprise needs	0	
	Choosing among design alternatives based on requirements, team performance, past experience, etc.	0	
	Ameliorating/ creating products	0	
	Monitoring, mitigating, and solving risks	0	
	Identifying stakeholder attitudes and preferences	0	
Common	Analyzing communication content and attitude	0	
	Information retrieval	1	PHPMyAdmin
	Prioritizing tasks	1	Teamwork
	Writing and presenting	0	
	Seeking help	0	
	Acquiring knowledge and learning tools and good practices	0	
	Social interaction	0	

- *ML*-2 (automated): the tool performs processing under control and monitoring of the engineer. In specific situations, the engineer may need to intervene or approve the tool's decision.
- *ML*-3 (delegated): the tool autonomously performs processing with no control of the engineer. The latter may check the tool's performance.

It is worth to note here that the literature reported a few other rating models of CASE tools, which are not relevant however, because they are not based on a cognitive perspective. In [2], a rating model is elaborated based on the COCOMO effort multipliers. In [3], a model is proposed to rate CASE tools according to how long they are available on the market.

Now that we identified maturity levels, we can analyze the maturity of the existing CASE tools. **Table 2** contains 198 CASE tools, which were analyzed based on the maturity model. This list was obtained from [4]-[6] after excluding tools that are no longer available on the Web and those which are irrelevant. For every task in **Table 1**, the corresponding set of CASE tools of **Table 2** has been identified. Then, the *ML* of the most mature among these tools was assigned to the task in **Table 1** (third column). If there are more than one tool with the highest *ML*, the fourth column in **Table 1** contains only one among them as example. Obviously, no CASE tool corresponds to *ML*-0. We could not, unfortunately, present the mapping between **Table 1** and **Table 2** because of the lack of space.

Considering *ML*-3 as the ideal level for cognitive CASE tools, the *maturity degree* (*MD*) of every set of competencies $A \in \{\text{Requirements, Programming, Testing, Managing, Common}\}$ can be calculated using **Table 1** and Equation (1):

$$MD(A) = \frac{\sum_{i=1}^{n} ML_i}{3n} \quad (1)$$

where n is the number of tasks and competencies under A, and ML_i is the ML of the task or competency i. The cognitive gap can then be derived using Equation (2):

$$CG(A) = 1 - MD(A) \quad (2)$$

The *MD* and the *CG* of Software Engineering as whole discipline can be calculated in the same way by considering *n* as the number of all the tasks and competencies in **Table 1**. **Table 3** contains the results. For example, 88% of the programming tools belong to *ML*-1 and 13% of the testing tools are at *ML*-2. Only 2% of all the CASE tools belong to *ML*-3 (last line in **Table 3**), and 6% of all the CASE tools are at *ML*-2. Requirement en-

Table 2. CASE tools.

	CASE Tool's Name		CASE Tool's Name		CASE Tool's Name
1	Acceleo	67	FuJaba	133	Psoda
2	Acclaro DFSS	68	G-MARC	134	Pylot
3	AceProject	69	Grinder	135	Qengine
4	Agilej	70	HP Fortify	136	QFDcapture
5	Agile Load	71	HP Openview	137	QPack
6	Aligned Elements	72	HTTPDebugger	138	Qtest
7	AllFusion	73	IBM Tealeaf	139	Ranorex
8	Amateras	74	IBM Workplace	140	RaQuest
9	Ameos	75	IBM-Rational	141	Rational DOORS
10	Android Testkit	76	Ideogramic	142	ReqMan
11	AnthillPro	77	Imagix 4D	143	Reqtify
12	AnyLogic	78	Innovator	144	Requirement Composer
13	AnyStates	79	inteGREAT	145	Requisite Pro
14	Aonix	80	IntelliUML	146	RMTrak
15	Appium	81	IRqA	147	Robotium
16	AppViewWeb	82	iUML	148	Rommana
17	ARCAD	83	J2U	149	RTIME
18	Architect	84	Janova	150	RW-UML
19	Architect	85	Jcrawler	151	Sandstorm
20	ArcStyler	86	Jdeveloper	152	Scenario Plus
21	Argo/UML	87	Jkool	153	SDE
22	Aris	88	Jsequence	154	SDMetrics
23	Artisan	89	Jubala	155	Selenium
24	Artiso	90	jUCMNav	156	Siege
25	AsiTrack	91	Jude	157	Silverrun
26	Astade	92	Jvision	158	Site Check
27	Avalanche	93	Katie	159	Skipfish
28	Avenqo PEP	94	Konesa	160	Soasta
29	Avignon	95	Leap	161	Sparx
30	Blazemeter	96	Load Lite	162	Spectrum
31	Blueprint	97	Load 2 Test	163	Speed Tracer
32	Bontq	98	Loadea	164	SpeeDEV
33	BOUML	99	Load Intelligence	165	Spira Team
34	Bridge Point	100	Loadster	166	Stress Tester
35	Bright	101	LoadStorm	167	Tau UML

Continued

36	BugImpact	102	LoadTracer	168	Teamwork
37	Bugzilla	103	LoadUI	169	Tellurium
38	CA Erwin	104	LoadZen	170	TestArchitect
39	Caliber	105	MacA&D/WinA&D	171	TestCafe
40	Cameo Req+	106	Mantis	172	TestTrack
41	CASE Spec	107	*MDE*	173	Together
42	Center Code	108	MeanPath	174	TopTeam Analyst
43	CloudForge	109	Mega Suite	175	Torture
44	Cognition Cockpit	110	Metabase	176	TraceCloud
45	CollabNet	111	MIA	177	TrackStudio
46	Concept Draw	112	Mink	178	TruWex
47	Coordinator	113	MKS	179	Tsung
48	Cradle	114	Monotone	180	Twist
49	CTS	115	MultiMechanize	181	Ubot
50	Cucumber	116	Neustar	182	Umbrello
51	Curl-Loader	117	Novosoft	183	UMT-QVT
52	CVS	118	NTOSpider	184	Via/Renaisance
53	Darcs	119	Nuxeo	185	VisibleThread
54	DBDesigner	120	Objecteering	186	Visio
55	Delphia	121	ObjectIF	187	Visual Studio
56	Describe	122	Omondo	188	vPerformer
57	Documentator	123	OpenSTA	189	WAPT
58	Documentum	124	OptimalJ	190	Watir
59	EctoSet	125	Oracle UnivCMS	191	WayPointer
60	Empirix	126	Outclip	192	WDK
61	ESS	127	Ozibug	193	Wilde
62	Estimator	128	PHPMyAdmin	194	Win A&D
63	eValid	129	Plastic	195	Wix
64	FactFinder	130	Polarion Requirements	196	WSOP
65	FL	131	Poseidon	197	Xeptance
66	Formoid	132	Prosa	198	yKAP

gineering's *MD* is 0.29. Software Engineering's *MD* is 0.22 and its *CG* is 0.78. This means that Software Engineering is 78% far from being fully delegated, in the sense of the here proposed maturity model. The least cognitive gap is scored by programming (0.58) then design (0.67).

4. Towards a Cognitive Software Engineering

4.1. Research Plan

The research question for the future is: how to reduce the cognitive gap of Software Engineering to 0? In other

Table 3. Maturity measures.

Phase	*ML*-3	*ML*-2	*ML*-1	*ML*-0	*MD*	*CG*
Requirements	0.00	0.29	0.29	0.43	0.29	0.71
Design	0.00	0.00	1.00	0.00	0.33	0.67
Programming	0.13	0.00	0.88	0.00	0.42	0.58
Testing	0.00	0.13	0.50	0.38	0.25	0.75
Managing	0.00	0.00	0.33	0.67	0.11	0.89
Common	0.00	0.00	0.22	0.78	0.07	0.93
Soft. Eng.	0.02	0.06	0.49	0.43	0.22	0.78

words: how to build a CASE tool for every competency in **Table 1**, which corresponds to *ML*-3? This research question raises another one about the scientific foundation of this endeavor: do CASE tools need to be cognitive systems? We conjecture that they do. This leads us to agree on the definition of cognitive systems because there is no single and widely adopted one. On the website of the European Network for the Advancement of Artificial Cognitive Systems (euCognition) [7], the definition of cognition itself is considered as one among a list of scientific controversies in the field. Interestingly, the euCognition organized a survey on the definition of cognition systems. Thirty eight definitions from respondents were selected and made available on their website. One can easily distinguish between two kinds of definitions; let us call them *high cognition* and *low cognition*. High cognition definitions of cognitive systems give emphasis to higher processes like abstracting percepts, having self-awareness, and interacting with people exactly like human beings (believable characters). Low cognition definitions of cognitive systems are more pragmatic. For the current research, we adopt one of these definitions which we borrow from Jir Wiedermann: "an artificial cognitive system is [⋯] designed by people to realize a cognitive task. The aim of the design is to construct a system producing a behavior that is qualified by system's designers as a reasonable behavior performing the task at hand. What is a cognitive task depends on the designers." Based on this definition, Cognitive Software Engineering aims at building CASE tools as cognitive systems that successfully perform cognitive tasks of software professionals. The CASE tools' aim is not to absolutely incarnate higher cognitive processes like perception and self-awareness. The only objective is to be effective. It is the responsibility of future research to verify whether cognitive CASE tools require higher cognitive abilities. This claim is not with no foundation, however. The research in cognitive systems itself did not completely solve the dilemma of whether a cognitive system must incarnate cognitive processes like emotion or suffice it to be effective for the targeted task [8].

Based on this epistemological principle, we can now propose a research process for Cognitive Software Engineering:

- *Identify the cognitive model of the task*: which describes the required cognitive operations and skills along with the ontology, knowledge, information structures, rules, and work procedures, which are involved in accomplishing the target task.
- *Design and Implement*: how can tools assist engineers in performing the task? How can tools replace engineers in performing the task? (Re)design the tool and implement.
- *Validate*: observe and evaluate.
- *Improve*: if necessary, go to step 1.

In the second step of this research process, the researcher has to decide whether the research's aim is *ML*-3 or a lower level. For the first step, identifying the cognitive model of the task, the researcher needs a reference cognitive model which contains the cognitive operations that might be involved in the studied task. For this aim, we here propose a modified version of Bloom's cognitive model also known as Bloom's taxonomy [9], which is presented in **Table 4**. Our modification consists in adding the set of synthetic cognitive skills, which we derived during the analysis of the Software Engineering cognitive tasks in order to elaborate **Table 1**. They are synthetic because they employ other elementary skills (remembering, understanding, etc.). Here are short definitions of the main cognitive processes of Bloom's taxonomy:

- Remember: retrieving knowledge from memory.

Table 4. Modified bloom's taxonomy.

Cognitive Skill	Sub-Skill	Cognitive Skill	Sub-Skill
Remember	Recognizing	Evaluate	Checking
	Recalling		Critiquing
Understand	Interpreting	Create	Generating
	Exemplifying		Planning
	Classifying		Producing
	Summarizing	Synthetic	Writing
	Inferring		Negotiating
	Comparing		Diagnosing
	Explaining		Monitoring
Apply	Executing		Predicting
	Implementing		Collaborating
Analyze	Differentiating		
	Organizing		
	Attributing		

- Understand: meaning of oral, written, and graphic messages.
- Apply: using a given procedure for a given purpose.
- Analyze: breaking the whole into its parts and determining how the parts relate to each other and to the whole.
- Evaluate: judging based on criteria.
- Create: assembling parts together to create a new whole.

4.2. Preludes of Cognitive Software Engineering

A few CASE research works analyzed the human cognitive processes and tried to automate them. They can be considered as the preludes of Cognitive Software Engineering. Argo/UML [10] is a CASE tool that contains a set of concurrent threads called critics. The latter monitor the work of the UML designer and check the conformance to the UML diagrams with specified syntactical and semantic rules. If a critic detects a deviation from any of those rules, it advises the designer. Argo/UML corresponds to *ML*-1 because part of the cognitive design tasks is automated. What is interesting in Argo/UML is that it has been built after the analysis of the cognitive processes involved in design [10]. From the perspective of **Table 4**, the author of Argo/UML considered the following cognitive skills:

- Predicting: critics "know" that if the designer interrupts a design task, he/she will probably switch to an alternative design option because there might have been a blocking problem. In such a case, critics will try to help the designer pursue the initial task in order to avoid the costly mental context switching.
- Recalling: if the designer insists on switching to another design task, critics will record the current context in order for the designer to avoid forgetting incomplete sub-tasks or details.
- Classifying and inferring: critics suggest to the designer to switch to a task that needs minimal updates in the current mental context instead of tasks that need important mental changes.
- Recognizing and generating: critics recognize the fixation situations where the designer focuses too much on one design alternative. They then try to bring the attention to other alternatives.

[11] deals with the problem of program comprehension. A CASE tool is designed and prototyped based on an analysis of the cognitive processes involved in program comprehension. The program is presented to the developer in terms of a network whose nodes are software components. The developer can choose different levels of abstraction for the nodes. Compared with **Table 4**, here are the main cognitive skills automated by this work:

- Explaining: nodes that correspond to software components are annotated as they are created.

- Organizing: pieces of code are associated with corresponding nodes in the tree.
- Recalling and recognizing: the tool records the path that led to the current node.

4.3. Cognitive Software Engineering: Lakatosian or Kuhnian?

Cognitive Software Engineering brings a new research interest, which consists to consider the *cognizant subjects*, namely software professionals, instead of continuing to focus on the application of *patterns*: patterns of methodologies, processes, system models, programs, and artifacts. Highly influenced by Systems Engineering and Project Management, the focus on patterns tended so far to put the cognizant subject under the control of best practices, best patterns. This is clearly expressed in early definitions of Software Engineering. For example, [12] defines Software Engineering as "the disciplined development and evolution of software systems based upon a set of principles, technologies and processes."

Let us try to epistemologically situate the here preached move from Software Engineering to Cognitive Software Engineering. The latter comes from the need to reduce the cognitive burden for software professionals. In the Kuhnian vision [13], such a need may be considered as a crisis, which is favorable to the emergence of a new paradigm. The new paradigm then represents a discontinuity with the former paradigm. In the Lakatosian vision [13], the need is not a real crisis, and the resulting move is characteristic of young scientific theories which did not find a dominating paradigm yet. Such a move preserves the continuity. Based on the Lakatosian epistemology, Software Engineering can be portrayed as consisting of a conceptual *core* and a conceptual *belt*. The conceptual core encompasses the set of disciplines (requirements, design, etc.), processes, patterns, models, tools, artifacts, metrics, and roles. A Lakatosian conceptual belt (also called: protective belt) is the set of auxiliary hypotheses. Software Engineering's belt consists of the set of widely agreed knowledge, rules, and best practices; for example, the advantages of Object Oriented analysis and design, the *de facto* link between real-time systems and concurrent processes or threads, the benefits of using a configuration management or a bug tracking system, the advantages of code refactoring, etc. A *problem shift* in the sense of Lakatos is considered as a widening of the conceptual belt, which leads to the *evolution* of the *research program* without affecting its core.

We conjecture that Cognitive Software Engineering is a problem shift in the sense of Lakatos, not a new paradigm in the sense of Kuhn. In other words: it is a natural evolution, not a conceptual revolution. It is complementary to Software Engineering and represents a widening of the conceptual belt of Software Engineering through introducing the cognitive dimension as part of the scientific problematic.

5. Conclusion

The present article sketched a roadmap for the evolution of Software Engineering towards Cognitive Software Engineering, through integrating the cognitive dimension in the CASE research. As stated in Section 4.2, a very few research works tried to tailor CASE tools to the actual cognitive processes that are employed by software engineers during their daily tasks. As a direct consequence, at the best of our knowledge, there is no commercial CASE tool that fully performs software cognitive tasks. The evolution from the current CASE tools to cognitive ones constitutes the research object of Cognitive Software Engineering. The methodology of this evolution was the main motivation of the current article. A research framework has been proposed, which consists of: 1) a comprehensive list of Software Engineering tasks and skills; 2) a generic cognitive model and a research procedure to analyze software professionals' needs, and design and implement cognitive CASE tools for them; and 3) a maturity model and a formal method to measure the maturity of CASE tools. As a result, the cognitive gap between the current Software Engineering and the targeted Cognitive Software Engineering was estimated to be 78%. This number cannot pretend to be accurate, however, because it highly depends on our analysis of 198 CASE tools, which cannot be mistake-free. An epistemological analysis of the relation between Cognitive Software Engineering and Software Engineering has also been performed. This analysis shows that Cognitive Software Engineering is a natural and Lakatosian evolution of Software Engineering, not a paradigm revolution in the sense of Kuhn.

References

[1] Moreno, A.M., Sanchez-Segura, M.I., Medina-Dominguez, F. and Carvajal, L. (2012) Balancing Software Engineering

Education and Industrial Needs. *Journal of Systems and Software*, **85**, 1607-1620 http://dx.doi.org/10.1016/j.jss.2012.01.060

[2] Baik, J. (2000) The Effects of Case Tools on Software Development Effort. Ph.D. Thesis, University of Southern California.

[3] Mosley, V. (1992) How to Assess Tools Efficiently and Quantitatively. *IEEE Software*, **9**, 29-32. http://dx.doi.org/10.1109/52.136163

[4] Berdonosov, V. and Redkolis, E. (2011) TRIZ-Fractality of Computer-Aided Software Engineering Systems. *Procedia Engineering*, **9**, 199-213. http://dx.doi.org/10.1016/j.proeng.2011.03.112

[5] Gea, J.M.C., Nicolás, J., Alemán, L., Toval, A. and Vizcaíno, A. (2012) Requirements Engineering Tools: Capabilities, Survey and Assessment. *Information and Software Technology*, **54**, 1142-1157. http://dx.doi.org/10.1016/j.infsof.2012.04.005

[6] (2014) ObjectByDesign. http://www.objectsbydesign.com/tools/

[7] (2014) European Network for the Advancement of Artificial Cognitive Systems. www.eucognition.org

[8] Stork, H.G. (2012) Towards a Scientific Foundation for Engineering Cognitive Systems—A European Research Agenda, Its Rationale and Perspectives. *Biologically Inspired Cognitive Architectures*, **1**, 82-91. http://dx.doi.org/10.1016/j.bica.2012.04.002

[9] Krathwohl, D.R. (2002) A Revision of Bloom's Taxonomy: An Overview. *Theory into Practice*, **41**, 212-218. http://dx.doi.org/10.1016/S0164-1212(98)10055-9

[10] Robbins, J.E. and Redmiles, D. (2000) Cognitive Support, UML Adherence, and XMI Interchange in Argo/UML. *Information and Software Technology*, **42**, 82-92. http://dx.doi.org/10.1016/S0950-5849(99)00083-X

[11] Storey, M., Fracchia, F. and Muller, H.A. (1999) Cognitive Design Elements to Support the Construction of a Mental Model during Software Exploration. *Journal of Systems and Software*, **44**,171-185. http://dx.doi.org/10.1016/S0164-1212(98)10055-9

[12] Basili, V. (1992) The Experimental Paradigm in Software Engineering. *Lecture Notes in Computer Science*, **706**, 3-12.

[13] Wautelet, Y., Schinccus, C. and Kolp, M. (2008) A Modern Epistemological Reading of Agent Orientation. *International Journal of Intelligent Information Technologies*, **4**, 46-57. http://dx.doi.org/10.4018/jiit.2008070103