The Role of Semiotic Engineering in Software Engineering

Vahdat Abdelzad
School of Electrical Engineering and Computer Science, University of Ottawa
Ottawa, Ontario, Canada
v.abdelzad@uottawa.ca

Timothy C. Lethbridge
School of Electrical Engineering and Computer Science, University of Ottawa
Ottawa, Ontario, Canada
tcl@eecs.uottawa.ca

Mahmood Hosseini
Faculty of Science and Technology, Bournemouth University
Poole, United Kingdom
tcl@eecs.uottawa.ca

ABSTRACT

Semiotic engineering is based upon the semiotic theory of Human-Computer Interaction (HCI), which focuses on communication between designers and users. Semiotic engineering tries to improve users' interpretation through metacommunication and emphasizes that designers should play the role of legitimate interlocutors in interactive systems. On the other hand, there is a gap in software engineering on how to obtain systems specifications efficiently, how to create easy-to-understand and communicative models, and how to produce comprehensive modeling languages and development processes. In this paper, we explore several contributions of semiotic engineering to software engineering and discuss how the theory can facilitate the creation of comprehensive artifacts. We also discuss semiotic engineering for assessing and improving software modeling languages, in our case UML. We anticipate that our work would lead to the semiotic theory becoming recognized as a central theory driving software engineering research and practice.

CCS Concepts

•**Software and its engineering** → **Software creation and management;** *Software system models;* •**Human-centered computing** → HCI theory, concepts and models;

Keywords

Semiotic engineering; communication; software engineering; modeling; artifact; UML

1. INTRODUCTION

In software engineering, scientists concentrate on such issues as development approaches, modeling tools, and testing methods so as to produce high quality software systems [24]. In order to achieve this goal, researchers and industrial companies have been using various approaches, such as Model-Driven Software Development (MDSD) [8]. Unfortunately, the nature of *communication* among the multiple stakeholders involved in software engineering has received little attention. An example of this is restrictions on expressiveness imposed by notations in requirements engineering [6].

In current software engineering approaches, it is possible to find patterns and guidelines that aim at facilitating communication, but it is rare to find a concrete theory supporting them. Communication in software engineering is primarily undertaken through artifacts, where each artifact might be produced by one or several stakeholders and can be used by many other stakeholders. Improper or immature communication may result in severe consequences, such as extra cognitive work for developers, misunderstanding of requirements, and failed software systems.

In the domain of human computer interaction (HCI), various theories, e.g., distributed cognition [18] and activity theory [19], have been developed to address communication. However, one theory, named *semiotic engineering* [13], has a distinctive perspective. This theory concentrates on communication as its base concept. Indeed, semiotic engineering is a theory of HCI which focuses on how well producers of software artifacts communicate their intent to their consumers through user interface signs and patterns of interaction [13]. In other words, semiotic engineering consists of a powerful infrastructure for the purpose of studying communication and it provides concepts to assess and improve communication between producers and consumers. Therefore, this theory focuses on communication as an issue often forgotten by scientists in both HCI and software engineering. Semiotic engineering is, consequently, an eligible candidate theory to be applied to software engineering in order to manage the communication challenge.

The goal of this paper is to draw attention to the concept of communication in software engineering in a scientific way through the theory of semiotic engineering. Bringing either a theory or a solution based on a theory into a field like software engineering is not an easy task. Hence, we believe there needs to be considerable thought and research before such a theory can become influential and successful. However, there is a need to start somewhere, and explore which theories have the potential to be used in the field. This paper hence provides some preliminary thoughts about the role of semiotic engineering in software engineering and why it has the potential. It also discusses the application of semiotic engineering in software modeling languages, such as UML, as an example to express how the theory can provide challenging questions and trigger research to seek proper answers.

TOSE'16, May 16 2016, Austin, TX, USA

ISBN 978-1-4503-4174-5/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2897134.2897136

The rest of the paper is structured as follows. Section 2 covers key background necessary to understand semiotic engineering. In Section 3, we focus on several relevant research projects in order to investigate communication using semiotic engineering theory in software engineering, and we discuss their explicit and implicit contributions and drawbacks. In Section 4, we explore applications of semiotic engineering as a method to evaluate communication. We discuss how one application of semiotic engineering can be used to find usability challenges related to the Unified Modeling Language (UML) [4]. This, in turn, exposes why UML may have some communication issues in terms of education, acceptability among developers, and lack of communication between UML designers and UML developers (software designers) as their users. Finally, we present our conclusion and future work in Section 5.

2. SEMIOTIC ENGINEERING

Semiotics is about studying signs and sign processes as part of communication [1]. It covers semantic, syntactic, and pragmatic dimensions of signs. In the semantic and syntactic dimensions, semiotics explores the meaning and formal structure of signs respectively. Finally, it studies the relation between signs and sign-using agents in the pragmatic dimension.

Semiotic engineering was initially proposed as a semiotic approach to design user interface languages [13]. However, it has been evolved over years into a semiotic theory of HCI. The theory concentrates on two fundamental concepts named metacommunication and meaning. Metacommunication is all about "communication about the communication". In other words, it is the main process held in the designer-to-user communication and system-user communication. This point of view considers designers and users as "legitimate interlocutors" at interaction time. In the theory, top level communication is considered as a one-shot comprehensive message paraphrased as [12]:

"*Here is my understanding of who you are, what I've learned you want or need to do, in which preferred ways, and why. This is the system that I have therefore designed for you, and this is the way you can or should use it in order to fulfill a range of purposes that fall within this vision*"

The subject "I" in the above paragraph specifies the designer of the system (or artifact) and the subject "you" is the user of the artifact. The type and content of the message as sent by the designer is completely related to the context of design. For example, there can be a guide regarding how to perform the interaction with the artifact in an HCI context or description of elements inside the artifact in a software development context.

Meaning is considered to be a culturally-determined, constantly evolving process. As a result, there is no fixed target to be met, captured, and encoded. This arises from the fact that human meanings change in both predictable and unpredictable ways, just as human life evolves. It emphasizes that it is impossible to fully understand the users' meaning, but it is possible to capture the relevant parts and encode them in systems so as to enable communication with users. Indeed, because of this nature there is a need for metacommunication.

Semiotic engineering is supported by two qualitative evaluation methods named the semiotic inspection method (SIM) and the communicability evaluation method (CEM) in order to evaluate the quality of metacommunication [14]. These two methods have the capability to be used in the direction of how to detect problems, how to improve the metacommunication, and how to generate new knowledge. The methods emphasize communication and signification processes rather than cognitive processes, which are mostly used in HCI evaluation methods.

3. RELATED RESEARCH

In this section, we present a summary of related research and discuss the contributions and drawbacks of these approaches. Our objective is to explain how semiotic engineering could have a positive contribution to software engineering in different dimensions. This section is not an exhaustive study about the application and effects of semiotic engineering on software engineering. We have focused on research that covers a wide spectrum of phases and activities in software engineering and also can effectively express the combination between the two fields.

3.1 Communication in Computer-Supported Modeling

Computer-supported modeling (CSMod) tools help us to define system behavior and desired system properties. There is a need for different kinds of communication with these tools in order to achieve software development goals. In this subsection, we explore how semiotic engineering could help researchers to evaluate a CSMod tool and offer some ideas about how to improve the communication.

Ferreira et al. [16] have combined and applied Semiotic Inspection Modeling (SIM) [14], Cognitive Dimensions of Notations (CDN) Framework [9], and Discourse Analysis (DA) [17] to ARIS Express (AE) [2], in modeling tasks with Business Process Modeling Notation (BPMN) [3], in order to analyze the tool from an HCI perspective and understand how communication is performed in software modeling. Indeed, they have focused on two dimensions:

1. how modeling notations respond to the expressive needs of model builders, and
2. how the context of communication is made available to the model builders.

The results suggest that CSMod design tools can be evolved in relatively unexplored directions, helping users (i.e., modelers) to gain greater awareness of the communication-through-models process. The results also show that although there is a large amount of documentation available for AE (in the form of tutorials, videos, manuals, etc.), when it comes to operation, the documentation is not as helpful as one would expect. AE delivers constraints of business modeling to users while it could have provided task-related help for them.

The following are some specific areas the paper highlights where investigation about communication through models could help to improve the tool. The evidence for these recommendations was generated by empirical observation and discourse analysis.

First, Ferreira et al. [16] determined that defining the *purpose of models* (the builder's intent) and the *targeted consumers* are two important challenges. Second, the evidence reveals that there should be a protocol between modelers and users in order to define which elements should be used or not, when and why. Lack of this protocol may raise a

cognitive issue called *diffuseness*, which is the complexity or verbosity of the notation in expressing meanings. Diffuseness has a negative impact on the completion of tasks. Third, there is a lack of *closeness in the mapping* of the representation to the domain; this is exemplified by icons that do not have clear meanings, forcing users to search for extra material in order to understand them. Finally, there is the issue of *secondary notation*, which is the ability to use notations beyond the formal syntax for expressing information or meaning. Neither AE nor BPMN provides such a notation. However, the availability of secondary notation has a positive effect on the completion of tasks.

Another cognitive dimension in AE which has a positive impact on the achievement of the task is *visibility*, which is the ability to view all components simultaneously, or two or more related components side by side at the same time. This CDN is achieved when AE allows users to choose different but related elements while they try to use one of them. It was also noticed that AE interface design supports model builders better than model readers.

In the domain of communication through models, Ferreira et al. [16] expressed that there are mismatches between the user profile that AE supposedly targets (occasional users and beginners) and the one that emerges from an analysis of emission and reception of its designers' message. It was also shown that designers apparently believe that it suffices to support the expression of communication and the interpretation will take care of itself. This is one of the important challenges in communication. The research concludes that if one wishes to discover the power of communication through models, a combination of semiotic, cognitive, and discourse analysis methods should be investigated. Together, not only can they tell us about how the CSMod design message is composed and how it affects the users as they build, edit or read models, but also they inform us about the cognitive challenge associated with the supported notations.

In our point of view, the significant part of the research is to construct a protocol for communication among models. This protocol could include social protocols as a good strategy to overcome representation limitations. This is really important because novices use the social protocol for learning the meaning of new notations and intermediate ones use it when they are challenged by several notations with different meanings. Consequently, in order to discover more issues about AE and BPMN, different levels of users (e.g., beginners, intermediates, and experts) could be considered and then explored separately, and various issues could be classified for different user levels. Furthermore, general issues that could happen to all users could be identified.

In the research, it would be possible to consider the theory of ecology [25] in order to know whether the level of abstraction for AE and BPMN is proper or not. By considering the level of abstraction, it would be clear which parts of the modeling need social protocols and which ones need technological protocols. In addition, it would reveal which issues are related to which levels. A mapping could also be created between user levels and abstraction levels in order to have more concrete and more practical findings.

Finally, the result of the research could be concretized by getting more feedback from users, e.g., by asking questions such as how they would like to tackle issues in each case. One good question which ought to be asked of users is whether they would like to model using a particular tool or modeling language. Answers to this question would reveal the impact rate of the issues on human behaviors in accepting a modeling language or tool. This is important because although human expectation in tools can only be satisfied, one can still identify problematic features and try to avoid them altogether.

3.2 Communication in Software Artifacts

In the process of software development, lots of artifacts are produced and used by stakeholders. These artifacts necessitate communication between producers and consumers, which needs to be studied. While it is possible to find guidelines for this purpose [20], these guidelines cannot ensure the suitability and helpfulness of communication. In this subsection, we look at a research project that explores communication between Application Programming Interface (API) designers and programmers.

In the research done by Afonso et al. [5], API is considered as an artifact mediating and easing the communication process between designers and programmers. Communication between APIs and programmers is evaluated based upon a combined semiotic and cognitive method. Furthermore, some tools and techniques are identified which help designers to accomplish the communication task.

Programmers need to realize the concepts and the design behind the interfaces available in order to use them effectively. This imposes a considerable amount of cognitive load on programmers, depending on the abstractions involved and the design of the artifacts provided. From a human-centric perspective, we may consider that a communication process takes place among programmers, mediated by the software artifacts involved. If this communication is not satisfactory, defects related to the incorrect use of APIs or to the misinterpretation of its design will arise in final systems. Therefore, designers need to provide necessary communication information through artifacts to decrease these kinds of defects.

The most common form of API specification is the combination of its syntactic (e.g., signatures) and semantic (e.g., behavior) elements written in a formal and natural language respectively. This form limits the designers' options to be "present" at the interaction time to provide more dynamic information to programmers. According to this limitation, environments which provide runtime monitoring and behavioral specification are considered to be useful because designers will have more opportunity to communicate with programmers. Contracts [21] are a good example for this purpose.

Furthermore, from a cognitive perspective, these environments or tools have an impact on the programmers' workload, since they provide a more precise description of the API behavior than the textual documentation, helping programmers to understand the causes of possible errors by giving them immediate feedback related to API misuses. Another point which can be achieved by behavioral specification languages is a higher expressiveness to describe a software artifact, allowing the use of tools, such as model checkers [7], to validate the specification.

It is furthermore determined by Afonso et al. [5] that the greatest focus of API specification is in syntactic and behavioral dimensions, and there is not enough attention paid to synchronization and quality of service. Communication in terms of synchronization is a valuable resource in express-

ing the designer's intents, as they offer a formal definition of the allowed sequence of operations. The quality-of-service dimension opens the possibility of specifying non-functional aspects of a software artifact that are more related to the execution environment or the precision of the results of the computation being carried out. This dimension offers designers an opportunity to specify the limitations or requirements of an API in terms of its execution environment.

From a semiotic engineering perspective, the main signs used by API designers in order to send their message to users are method signatures, return values for methods and other related operations, such as insertion and removal from collections (dynamic signs), and the textual description. However, there can be some extensions in order to make this communication more effective, e.g., better code examples, methods to test consistency, and formal specifications. From a cognitive perspective, it might be possible to provide interesting insights regarding this particular design, e.g., a hidden dependency between classes in the API, viscosity, and premature commitments, as these are not obvious at first, especially to novices.

Many defects of software development, recognized by semiotic engineering, are shown by Afonso et al. [5] that are due to poor communication among developers (designers and programmers). However, we believe that the most important result of the research, not clear at first glance, is their categorization of several communication problems in software development, each of which can be resolved by different theories of HCI. Furthermore, the research attempts to show that semiotic engineering can be considered as a powerful theory in the domain of interaction, which might be between two humans or between a computer and a human. It may be understood that all artifacts in software development can be considered as mediation between their creators and users. Therefore, there should be comprehensive metacommunication strategies to be used by designers so as to provide all stakeholders with needed information in artifacts.

The research conducted by Afonso et al. [5] is a start for future research and it does not give more detailed information about how to create these kinds of metacommunication. Another thing worth mentioning is that semiotic engineering might not have a concrete solution for problems that it discovers. However, it has the potential to be extended into the domain of problem solving. For example, researchers working in the domain of software documentation and maintenance may use rich conceptual definitions of metacommunication so that they change the structure of the current format in the documentation. Moreover, this potential may also be used for changing the nature of graphical and textual modeling languages used for communication among stakeholders.

3.3 Communication in Better Description

HCI developers are responsible for creating suitable user interfaces, and software engineers develop software systems to cover the required functionality and all other necessary requirements. Both groups start their work from the stated requirements but with different purposes. This can pose a big communication challenge between these two groups when system-user interaction is poorly understood. Below, we discuss a research project that focuses on how a tool based upon semiotic engineering can bridge the gap.

Modeling Language for Interaction as Conversation (MoLIC) has been discussed in [10, 11, 22]. MoLIC is a modeling language for HCI based upon semiotic engineering, and is an extension to UML diagrams with the purpose of removing some existing ambiguities in models of software systems developed using UML. The ambiguities arise because UML does not have an acceptable coverage of user interaction modeling.

It is pointed out that user interaction diagrams should be considered as a blueprint of the system. Such a blueprint could be used as a reference point for global design decisions, and would be an additional resource for deriving both HCI and software engineering models. The blueprint can be enhanced by MoLIC because it adopts the HCI theory and provides us with an ontology for describing and evaluating relevant HCI phenomena, always keeping the focus on the quality of use of the proposed solution.

According to the proposal [10], modeling should be done after use case elicitation and specification. Then, class diagrams can be created or improved by detailed interactive information obtained from a MoLIC model. The advantages of using MoLIC in this part of the process is that no system decomposition needs to be made or revised before this step, and thus the cost of changes remains low. Furthermore, designers will be motivated to find and correct problems in these information sources, such as inconsistencies and incompleteness.

The paper reveals how theories in HCI can help software developers to build comprehensive models for software systems. On the other hand, it shows how it is possible to combine HCI and software modeling with each other. The research implicitly shows that the lack of good interaction modeling diagrams can damage communication among developers of software systems. This is possible because software developers need to be able to explore such models to understand the whole system.

In our point of view, enhancements could be made if the authors had created a mapping from MoLIC to the UML extensions mechanism, because MoLIC has a good theory background, but its technical structure is not strong enough to be chosen as a good combination for UML. The profile extension of UML could be used to cover MoLIC concepts. In that case, it may increase usability and also easy acceptance of the concept in software engineering.

3.4 Communication in Testing

Testing usability is a key task in both HCI and software engineering. Engineers utilize various techniques and criteria in this process. Comprehensive testing includes checking all requirements from HCI and software engineering perspectives. The research done by Schilling [26] looks at this challenge (how to test systems from both perspectives) by proposing a software development method inspired by semiotic engineering.

An Interactive System (IS) development method is proposed by Schilling [26] for performing usability tests in earlier stages of software development, based upon the integration of concepts from models used in usability, semiotics, and software engineering. Three major engineering phases are considered for this purpose. The first one, using methods from usability engineering, supports gathering information and verifying and validating user interfaces. In this phase, several user interface alternatives from user interface

requirements models should be derived. These models express the need, preference, and constraints of both users and clients, and are obtained from qualitative and quantitative research. Based on the obtained results, all user interfaces will be then evaluated. The second phase follows standard software engineering testing approaches, testing IS after the execution of the implementation and integration activities. The last phase is semiotic engineering, which tests IS usability with real users in the real context of use. This phase allows developers to test the interactivity and communicability between the user and system to investigate how the user interface affects users' activities and how they achieve their goals through the user interface.

Integration of models belonging to three engineering domains brings advantages that can be viewed from two perspectives. From the users' point of view, it can result in decreased learning time and increased user satisfaction. From the developers' point of view, it can be used to improve communication among developers to help them perform usability test tasks in an efficient manner by using the same vocabulary and artifacts.

Schilling's research [26] shows the usability of semiotic engineering in the software development process. By considering semiotic engineering explicitly as a final phase in usability testing, it is revealed that the theory can provide acceptable feedback on usability problems. On the other hand, the nature of the proposed process can yield a good sign of the application of semiotic engineering in the improvement of software development processes.

Furthermore, the lack of good metacommunication among different models leads to more time spent on developing a software system. The combination of various models in Schilling's research [26] is a kind of communication that provides automatic test generation. According to this simple proposal, we should extend the concept of communication in different ways to get maximum benefits from different models created during the software development and user interface design.

Schilling et al. [26] claimed good automatic test generation, but it cannot be observed in the data available in the paper. Moreover, the description for phases is rather abstract, causing the reader not to understand the exact advantages and disadvantages of the proposed process.

4. OUR PERSPECTIVE

In this section, we discuss how semiotic engineering may be used to evaluate and improve communication between producers and consumers in software engineering. We focus on some challenge in UML which might be either discovered or improved by having a semiotic engineering perspective. Indeed, the goal is to express how semiotic engineering can approach the challenges existing in software engineering.

4.1 Role of Semiotic Engineering

The focus of semiotic engineering is on communication, especially computer-mediated designer-user communication. It points out that rich communication should be provided by one-shot messages which designers give to their users through the media they produce. This concept is powerful because several things around us have at least a designer (producer) and a user (consumer), so the theory can be applied to several other cases as well. Therefore, software artifacts such as models can also be viewed as one of these cases.

Unfortunately, it is hard to find a theory in software engineering to aim at communication. This stimulates the question in our minds regarding how we can expect good communication among software artifacts while we do not know whether or not there are enough data, symbols, and structures in artifacts to facilitate such communication.

Software artifacts are created in the process of software development and their producers are goal-oriented. This means that they primarily attempt to satisfy software development requirements and pay little attention to items such as:

- how artifacts will be used in the future;
- how easy artifacts are to interpret;
- how artifacts will reveal their designers' hidden presumptions;
- how much cognitive work artifacts will put on the users

Therefore, it is necessary to adopt a theory that covers these questions by providing a method for evaluation and improvement. We can propose that there should be a method to evaluate software artifacts. This method will finally be extended to a concrete framework that allows developers to do tradeoff analysis. The core of the method should be prepared and covered by semiotic engineering theory. For example, there is a method in [16] used to evaluate CSMod tools and it is a combination of semiotic engineering and CDN. In the method, it is necessary to consider software engineering criteria to evaluate the effectiveness of artifacts for having rich communication. Since cognition is a characteristic of artifacts, the positive and negative effects of cognitive notations in software artifacts should also be involved. This should get more attention because measuring those effects may depend more on the context.

It can be seen how following the concept of communication and semiotic engineering provides us with questions and partial answers to get the final answer which can be a framework in this case.

4.2 UML and Semiotic Engineering

In order to figure out the potential relationship between UML and semiotic engineering, we focus on some questions that may be answered by it. Most of the questions are challenging and need to be explored to a considerable extent, so as to find more concrete answers. However, the questions show that UML needs to be rechecked based upon HCI theories, especially semiotic engineering. This rechecking should be done more in the direction of usability challenge.

In our discussion, UML is considered in two dimensions. The first dimension is about UML models as software artifacts whose producers are software designers, and whose consumers are software stakeholders. This dimension is supported to some extent by methodologies, but it is hard to find a concrete theory that clearly specifies the nature of these artifacts. The second dimension is about communication between model developers and UML itself. Indeed, in the second dimension, producers are UML designers (e.g., researchers who work on extending UML meta-model) and consumers are software engineers who use UML for software development. There is a gap in this dimension because communication between designers of UML and its users has not

been defined very well; at least we do not see such communication.

UML is designed and developed mainly by the Object Management Group (OMG). It is used in different areas and most of its users are developers. Developers need to communicate with UML tools so that they can model the target system. If there is communication between developers and tools, there should be a method or theory to support it in an appropriate way.

In the context of model communication, consumers are not typical information technology (IT) users; they are software developers. Typically, we talk about UML tools for providing better communication among developers using UML, but it might be possible to have some other factors, which play hidden roles (e.g., the nature of models or diagrams). Currently, there is a tendency in software research communities that UML has the necessary expressiveness for communication, but in practice UML is not being used in their projects [23] or the levels of regular usage of UML components are not as expected [15]. We think that one of the reasons for this issue can be due to communication issues. This can be clarified by the fact that it has been verified based on experiences, psychology, science, and engineering that modeling is beneficial, so MDSD is the right approach. Furthermore, developers believe in modeling but do not use UML. It should be pointed out that modeling can be textual and graphical, so the issue may not be just about notations and graphical elements used for UML.

We believe that UML evaluation should be separated from its tools and this can be achieved by using semiotic engineering. There are lots of tools that support UML, so the selection of tools for the study can affect the final results. The evaluation should be based upon concrete syntax, structure, and cognitive effects. If UML is evaluated based on tools, core communication challenge in the nature of UML cannot be found.

Another interesting subject that has been explored in HCI is whether reducing cognitive load has a positive effect on usability and learning. The designers of UML, we believe did not pay much attention to cognition, focusing instead on having strong structure and coverage. However, they should consider that UML models are created by users and are interpreted by computers and humans. Therefore, the cognitive dimension of UML should be studied and modified to enable better usability. The following are examples of topics that could easily be studied by semiotic engineering:

- What is the extent to which specific details should appear in class diagrams such as empty boxes when there are no attributes or methods to display, or mandatory type and visibility information?
- To what extent can specific diagrams, like state machines and class diagrams, be used together?
- What is the cognitive load of various notations?

In general, the theory can help UML designers to play their role as legitimate interlocutors.

As seen, following the theory challenges UML and somehow provides guidelines which can be investigated and applied to UML. This exposes the fact the semiotic theory has the potential to be applied to software modeling languages, but there is still a need for more studies to be done in order to make the theory available for software engineering in general.

5. CONCLUSION AND FUTURE WORK

In this paper, we explored research contributions of semiotic engineering to software engineering in general and to modeling in particular. We pointed out why the combination of semiotic engineering with different concepts of software engineering should be considered. We explored some the implicit and explicit contributions and drawbacks of the approach. Our key point is that semiotic engineering theory can be beneficial in software engineering because it focuses on communication, which is also central to the whole process in software engineering.

Furthermore, this paper proposed initial ideas about the use of semiotic engineering theory along with other theories as a method to evaluate and improve software artifacts as computer-mediated communication between producers and consumers. The paper discussed certain challenges of UML that can be explained with and explored by semiotic engineering. Although there is no concrete framework or theory proposed so far for this purpose, it shows how the semiotics perspective on the challenge of software engineering can open new thoughts and solutions.

A good direction for future work would be to obtain a more concrete interpretation about the use of semiotic engineering theory in software engineering. This can be done by exhaustive study of research interaction between semiotic engineering and software engineering. Another direction would be to create a concrete method for evaluating and improving software artifacts based on semiotic engineering, cognitive dimensions, and software engineering theories.

6. REFERENCES

[1] The science of communication studied through the interpretation of signs and symbols as they operate in various fields, esp. language. In *Oxford English Dictionary*. 2003.

[2] ARIS Express Free Modeling Software. http://www.ariscommunity.com/aris-express, 2016.

[3] Business Process Model and Notation (BPMN). http://www.omg.org/spec/BPMN/, 2016.

[4] Unified Modeling Language (UML)- Object Management Group. http://www.omg.org/spec/UML/, 2016.

[5] L. Afonso, R. Cerqueira, and C. de Souza. Evaluating application programming interfaces as communication artefacts. In *Psychology of Programming Interest Group Annual Conference*, pages 151–162, 2012.

[6] A. Al-Rawas and S. Easterbrook. Communication Problems in Requirements Engineering: A Field Study. In *Proceedings of the First Westminster Conference on Professional Awareness in Software Engineering, Royal Society, London*, 1996.

[7] C. Baier and J.-P. Katoen. *Principles Of Model Checking*. MIT Press, 2008.

[8] S. Beydeda, M. Book, and V. Gruhn. *Model-Driven Software Development*. Heidelberg: Springer, 2005.

[9] A. Blackwell and T. Green. Notational systems-the cognitive dimensions of notations framework. In *HCI Models, Theories, and Frameworks: Toward a*

Multidisciplinary Science, pages 103– 134, San Francisco, 2003.

[10] M. de Paula and S. Barbosa. Investigating the role of a model-based boundary object in facilitating the communication between interaction designers and software engineers. In *6th international conference on Task models and diagrams for user interface design*, pages 273–278, 2007.

[11] M. G. de Paula, S. D. J. Barbosa, and C. J. P. de Lucena. Conveying human-computer interaction concerns to software engineers through an interaction model. In *Latin American conference on Human-computer interaction*, pages 109–119, New York, New York, USA, 2005. ACM Press.

[12] C. de Souza. Semiotic Engineering - A New Paradigm for Designing Interactive Systems. *The Past and Future of Information Systems: 1976-2006 and Beyond SE - 21*, 214:231–242, 2006.

[13] C. S. de Souza. *The semiotic engineering of human-computer interaction*. The MIT Press, Cambridge, Massachusetts, London, England, 2005.

[14] C. S. de Souza and C. F. Leitão. *Semiotic Engineering Methods for Scientific Research in HCI*. Princeton: NJ. Morgan & Claypool, jan 2009.

[15] B. Dobing and J. Parsons. How UML is used. *Communications of the ACM*, 49(5):109–113, 2006.

[16] J. J. Ferreira and C. S. D. Souza. Communicating ideas in computer-supported modeling tasks : A case study with BPMN. In *HCI International 2013 - Human-Computer Interaction*, pages 320–329, 2013.

[17] J. P. Gee. *An Introduction to Discourse Analysis: Theory and Method*. London: Routledge, 2005.

[18] E. Hutchins. *Cognition in the Wild*. Cambridge, MA:MIT Press, 1995.

[19] K. Kuutti. Activity theory as a potential framework for human-computer interaction research. *Context and Consciousness: Activity theory and human-computer interaction, Cambridge: MIT Press*, pages 14–44, 1995.

[20] I. R. McChesney and S. Gallagher. Communication and co-ordination practices in software engineering projects. *Information and Software Technology*, 46(7):473–489, jun 2004.

[21] B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, 1992.

[22] M. G. D. Paula, S. D. J. Barbosa, C. J. P. D. Lucena, D. D. Informática, and R. M. D. S. Vicente. Relating Human-Computer Interaction and Software Engineering Concerns : Towards Extending UML Through an Interaction Modeling Language. In *Closing the Gaps: Software Engineering and Human-Computer Interaction*, pages 40–46, 2003.

[23] M. Petre. UML in practice. In *Proceedings - International Conference on Software Engineering*, pages 722–731, Piscataway, NJ, USA, may 2013. IEEE Press.

[24] R. E. D. F. Pierre Bourque. *Guide to the Software Engineering Body of Knowledge Version 3.0 (SWEBOK Guide V3.0)*. IEEE Computer Society, 2014.

[25] Samuel M. Scheiner and Michael R. Willig. *The Theory of Ecology*. University of Chicago Press, 2011.

[26] A. Schilling, K. Madeira, P. Donegan, K. Sousa, E. Furtado, and V. Furtado. An integrated method for designing user interfaces based on tests. In *Proceedings of the 1st International Workshop on Advances in Model-based Testing*, A-MOST '05, pages 1–5, New York, NY, USA, 2005. ACM.