Directives of Communicability for Software Models as Boundary Objects in Software Development

Adriana Lopes[1], Tayana Conte[1], Clarisse Sieckenius de Souza[2]

[1] USES – Grupo de Usabilidade e Engenharia de Software
Universidade Federal do Amazonas (UFAM) Manaus, AM – Brazil
{adriana, tayana}@ufam.edu.br

[2] Semiotic Engineering Research Group
PUC-Rio, Rio de Janeiro, RJ – Brazil.
clarisse@inf.puc-rio.br

USES Technical Report
RT-USES-2018-0004
May, 2018

Institute of Computing (IComp)
Federal University of Amazonas (UFAM)
Manaus, Amazonas 69077-000

ABSTRACT

This technical report presents the Directives of Communicability (DCs) proposed to improve the quality of communication among software development team members. The development of DCs was based on Grice's Cooperative Principle and Semiotic Engineering, theories that investigate different ways of communication. We performed two studies to evaluate DCs, and the results showed that it supports the development of software models with information more clearly expressed by their producers. This characteristic improves the communication among software development team members. The results of the study provided initial evidence regarding the feasibility of DCs to mitigating risks in the communication. From this technical report, we believe that DCs can benefit software development team regarding the risks for effective communication through software models.

1. INTRODUCTION

Managing communication among software development team members is a challenge in Software Engineering [1]. With model communication, specifically, there are several issues because, among other things, models may be inconsistent or ambiguous [2]. Misinterpretations by model consumers are communication failures, which can introduce incorrect information in software development activities, and subsequently lead to defects that lie at the origin of rework or quality loss [2].

The risks of miscommunication through software models are related to model content and to how content is expressed by its producer (model form). For example, in terms of content, a model can provide more or less information than necessary. Likewise, in terms of form, a model can be expressed using ambiguous representations. Both problems may ultimately impair the model consumer's understanding.

Our long-term research project aims to find out how to reduce failures in communication among software development team members through models as boundary objects. On our path to answering this question, we created the Directives of Communicability (DCs) and carried out a two-phase study to evaluate their impact. Communicability in this context refers to the boundary object's ability to convey to its consumers the solution conceived by its producers. DCs have, thus, been designed to improve the quality of communication between model producers and consumers in software development activities.

In this research, we follow a Human-Centered Computing (HCC) perspective, where machines, humans and application domains are investigated together [3]. Moreover, we use Semiotic Engineering [4-5], a Human-Computer Interaction theory that has recently been extended to account for topics in HCC. It investigates the forms of communication that take place at development and use time, and also the relations between them. Semiotic Engineering proponents [5] claim that the Grice's Cooperative Principle [6] can be helpful for the analysis of effective and efficient communication in such settings. Therefore, we have grounded the design of DCs in Semiotic Engineering and Grice's Cooperative Principle.

The remainder of this technical report is organized as follows: Section 2 presents background. Section 3 presents the DCs (latest version so far). The Appendix A presents the version of DCs that can be used by professionals and researchers.

2. BACKGROUND

In this section, we present the background on Semiotic Engineering and Grice's Cooperative Principle, both used in this work.

2.1 Semiotic Engineering

Semiotic Engineering theory [4], characterizes user-system interaction as a particular case of human-mediated systems communication. Systems are considered *metacommunication* artifacts in Semiotic Engineering, i.e., artifacts that communicate a message from the designer to users about how they can or should communicate with the system to do what they want. The content of the *metacommunication* message, or *metamessage*, can be paraphrased in the following template:

"***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***".

Semiotic Engineering extended its original perspective to a Human-Centered Computing (HCC) perspective. HCC is a field of research that aims to understand human behavior by integrating technologies in social and cultural contexts [3]. This contribution is related to the set of conceptual and methodological tools called SigniFYI (Signs For Your Interpretation) [5]. The SigniFYI Suite assists in the investigation of meanings in a software during the development process and in the communication between producers and consumers of software.

2.2 Grice's Cooperative Principle

The Cooperative Principle proposed by Paul Grice [6] to characterize the logic of conversation can be used to characterize the communication between model producers and consumers as well. According to Grice, productive conversation (communication) depends on the observation of reciprocal cooperation, which is established by four maxims:

Quantity - Make your contribution as informative as necessary, and no more than necessary;

Quality - Try to make your contribution true. Do not say what you believe to be false and do not say something that you do not have adequate evidence of;

Relation - Be relevant, that is, do not introduce issues that do not come to the case under discussion; and

Manner - Be clear, brief and organized with your input. Avoid obscurity of expression, ambiguity.

Breaking one or more of these maxims may lead to communication failure. However, an adequate use of Grice's maxims involves the concept of implicature, that is, information that can be inferred from statements. Conventional implicatures can be inferred from the conventional meaning of word. But there are also conversational implicatures, that is, inferences that can be drawn from participants of a given conversational context in order to fulfill certain gaps and omissions in order to (re)establish coherence and consistency in communication. Therefore, unlike conventional implicatures, conversational implicatures cannot be resolved by invoking the usual meaning of information represented in communication and require different kinds of inferences.

3. DIRECTIVES OF COMMUNICABILITY FOR SOFTWARE MODELS

The DCs were developed with expressions that can characterize the producer's communication to consumers. To support the use of DCs, we based on the Semiotic Engineering metacommunication template to help producers think about consumers before model development. We adapted the original template of this theory to:

"***Here is my understanding, as a producer of the model, of who is the consumer*** (to whom the producer is designing the model), ***what I've learned about what you need to do in system development*** (about what should be addressed in the model). ***This is the solution of the system that I designed for you to carry out your activities***".

Researchers and professionals interested in using the DCs can obtain them in Appendix A. Below we present each DCs. Table 1 shows the process to support the use of the DCs. The DCs are:

- "**Say the truth!**" - **DC1**: Use true information. Do not use information that affects the quality of the model (maxim of Quality).

- "**Say what is needed and no more than necessary**" - **DC2**: Use the necessary content in the template. Do not use unnecessary content in the model (maxim of Quantity).

- "**Say it logically**" **- DC3**: Organize the information in the model consistently (maxim of Relation).

- **"Say it clearly"** - **DC4**: Organize the information in the model clearly (maxim of Manner).

Table 1. Process to Support the Use of the DCs

S#	Description of use
Step 1	**DC1** (maxim of Quality).
Step 2	**DC2** (maxim of Quantity), with the following combinations: **D2.1** – Regarding the information that is necessary and no more than necessary in the model, do not include information that affects the quality of the model **(**Quantity and Quality**)**.
Step 3	**DC3** (maxim of Relation), with the following combinations: **D3.1 -** In the case where incomplete or extra information is relevant in the model, justify (Relation and Quantity). **D3.2** – In the case where false information is relevant, justify (Relation and Quality).
Step 4	**DC4** (maxim of Manner), with the following combinations: **D4.1** - Keep the conciseness, without the sacrifice of coherence (Manner and Relation). **D4.2 -** Keep the conciseness, without the sacrifice of what is needed (Manner and Quantity). **D4.3 -** Keep the conciseness, without sacrificing quality (Manner and Quality).

Before using DCs for building software models, some factors should be considered by producers so that their intention to communicate through the models are comprehended.

(i) Model as a communication channel - *the producer must analyze whether, through the model, the contents for which he wishes to communicate through the models, can be represented*. For instance, if the producer wishes to communicate through a model how the users can perform a task and what is the system's response regarding such tasks, he should not choose the UML class diagram for that purpose.

(ii) Context of use of the model - *the producer should analyze which models are suitable for the software organization as a means of communication*. For instance, if the producer wants to use a model which is not known by the software development team, this may not be adequate to support the communication between their producers and consumers.

To support the use of DCs, we based on the Semiotic Engineering metacommunication template to help producers think about consumers before model development. We adapted the original template of this theory to:

"***Here is my understanding, as a producer of the model, of who is the consumer*** (to whom the producer is designing the model), ***what I've learned about what you need to do in system development*** (about what should be addressed in the model). ***This is the solution of the system that I designed for you to carry out your activities***".

Based on this, we developed a proposal that supports the reflection of the producers on the modeling. This proposal has the following questions:

(i) For whom is the model being designed? – *"Can the content of the model be comprehended so that the consumer accomplishes its objectives?"* - to support the producer to reflect whether the information in the model can be understood by all involved, such as developers and managers, or only developers;

(ii) What is being addressed in the model? – *"What content should be addressed about the system's problem/solution domain in the model?"* - in order to encourage the producer to reflect on the content that he wishes to be comprehended from the model, such as the tasks that a user can perform on the system.

The Figure 1 presents an example about factors that should be considered by producers, the proposal that supports the reflection of the producers on the modeling and the DCs.

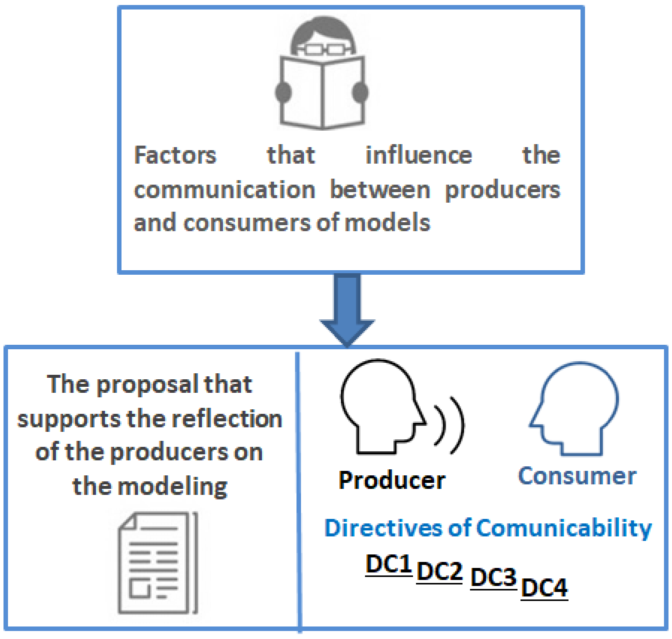
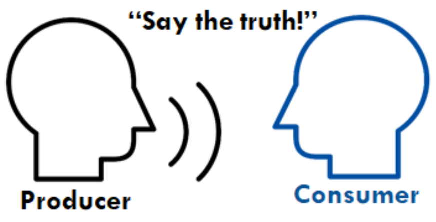
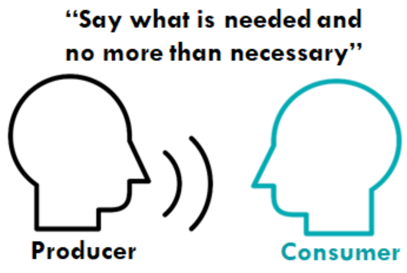
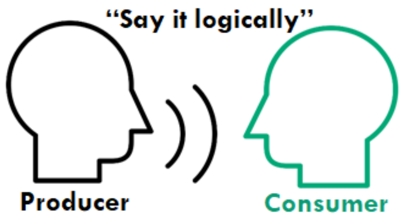
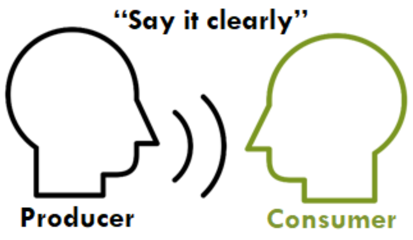

Factors that influence the communication between producers and consumers of models
The proposal that supports the reflection of the producers on the modeling
Producer
Consumer
Directives of Comunicability
DC1 DC2 DC3 DC4

Figure 1: Example of use of the DCs.

4. APPENDIX A

DIRECTIVES OF COMMUNICABILITY FOR SOFTWARE MODELS	
	**DC1**: Use true information. Do not use information that affects the quality of the model.
 "Say what is needed and no more than necessary" Producer Consumer	**DC2**: Use the necessary content in the template. Do not use unnecessary content in the model. **D2.1** – Regarding the information that is necessary and no more than necessary in the model, do not include information that affects the quality of the model.
	**DC3**: Organize the information in the model consistently. **D3.1** - In the case where incomplete or extra information is relevant in the model, justify. **D3.2** – In the case where false information is relevant, justify.
	**DC4**: Organize the information in the model clearly. **D4.1** - Keep the conciseness, without the sacrifice of coherence. **D4.2** - Keep the conciseness, without the sacrifice of what is needed. **D4.3** - Keep the conciseness, without sacrificing quality.

REFERENCES

1. A.H. Reed, and L.V. Knight, 2010. Effect of a Virtual Project Team Environment on Communication-Related Project Risk. International Journal of Project Management, 28 (5), 422–427.

2. R. M. De Mello, E. N. Teixeira, M. Schots, C. M. L. Werner, and G. H. Travassos. 2014. Verification of Software Product Line Artefacts: A Checklist to Support Feature Model Inspections. Journal of Universal Computer Science, 20(5), 720-745.

3. N. Sebe. 2010. Human-Centered Computing. In Nakashima, H., Aghajan, H., & Augusto, J (Eds.), Handbook of ambient intelligence and smart environments, 349–370. DOI: 10.1007/978-0-387-93808-0_13.

4. C. S. De Souza. 2005. The Semiotic Engineering of Human-Computer Interaction (Acting with Technology) (1st ed.). The MIT Press.

5. C. S. de Souza, R. F. de G. Cerqueira, L. M. Afonso, R. R. de M. Brandão, and J. S. J. Ferreira. 2016. Software Developers as Users: Semiotic Investigations in Human-Centered Software Development (1st ed.). Springer International Publishing Switzerland. DOI 10.1007/978-3-319-42831-4.

6. Grice H. P. 1975. Logic and Conversation. Syntax and Semantics 3: Speech arts, ed. Peter Cole and Jerry Morgan, 41–58.