ComD2: Family of Techniques for Inspecting Defects in Models that Affecting Team Communication

Adriana Lopes[1], Ursula Campos[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, usc, 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-2018-0003
March, 2018

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

ABSTRACT

This technical report presents a family of techniques for inspecting defects that affecting team communication, called ComD2 (Communication between Designers and Developers). The ComD2 family was developed based on theories that investigate different ways of communication. We have initially developed three specific inspection techniques for UML class diagrams, activity diagrams, and state machine diagrams. The techniques were developed for these models because they are among the most frequently used in the industry. We present the development of the verification items in the ComD2 family of techniques. From this technical report, we believe that other specific techniques can be created for the different models. Furthermore, we present a material to evaluate the feasibility of inspection techniques.

1. INTRODUCTION

According to Reed and Knight [1], effective communication is one of the most critical components of working in software teams. In software development, the communication is carried out through face-to-face discussions in co-located or distributed teams [2], besides the support offered by tools [3]. Software models are also used as means of communication in software development teams [4]. In this paper, we explore the communication of software development teams through software models.

Software models that support the communication in different domains can be considered boundary objects. Boundary objects are used for different purposes and in different domains while maintaining their authenticity [5]. The term boundary object comes from the use of objects that facilitate the sharing of information across linguistic, cultural, or knowledge boundaries, such as the communication between a software development team and its client.

Communication failures from software models can come from information that is not clearly expressed by their producers (people who created the models). Thus, other members of the development team (i.e. consumers, who comprehend the models for the creation of other artifacts) may have different interpretations of the ones intended by the producers. Different interpretations can introduce incorrect information into other artifacts and generate various defects during the production of software; such as the omission of some necessary information or the vague definition of information, thus allowing multiple interpretations [6].

We have developed a family of techniques called ComD2 (Communication between Designer and Developers). The purpose of the ComD2 family is to support the identification of defects that affect the team communication, i.e. the communication of the designer1 to the developers at development time. The collaboration between designer and developers is one of the factors for the success of software development [8]. We have initially developed three specific inspection techniques for UML class diagrams, activity diagrams, and state machine diagrams. The techniques were developed for these models because they are among the most frequently used in the industry [9].

The ComD2 family of techniques was developed based on theories related to Human-Centered Computing (HCC), a field of research that integrates theories and methodologies in research on machines, human and application domains [10]. In particular, we used Semiotic Engineering, a theory originally proposed for Human-Computer Interaction [11][12], which has been extended to account for HCC, investigating different forms of communication through and about software, both during development (between producers and consumers of software development artifacts) and at use time (between software producers and end users,

[1] We use the term *designer* for the Software Designer, also called Information Architect, i.e., the professionals involved in designing the software solution.

through systems interfaces). De Souza et al. [12] propose that Grice's Cooperative Principle [13] can be used to assess the effectiveness and efficiency of communication achieved through software products (or representations thereof). Thus, we also adopted this Gricean principle as a theoretical basis for the development of the ComD2 family.

The verification items of the ComD2 family help practitioners classify different defects that are found in software models [14] and detect which dimension(s) of model representation associated with the defect can lie at the origin of potential miscommunication. The employed dimensions of representation are Syntactic (relationship between model and the modeling language), Semantic (relationship of the model with the problem domain) and Pragmatic (relationship of the model with the stakeholders) [15].

The remainder of this technical report is organized as follows: Section 2 presents the verification items base for ComD2 family. Section 3, 4 and 5 presents the three specific inspection techniques for UML class diagrams, activity diagrams, and state machine diagrams. Finally, Section 6 presents the forms for the participants to report the identified discrepancies and questionnaires for evaluation of ComD2 techniques.

2. VERIFICATION ITEMS BASE FOR COMD2 FAMILY

2.1 Background

2.1.1 Grice's Cooperative Principle

The Cooperative Principle proposed by Paul Grice [15] 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 ommissions 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 informations represented in communication and require different kinds of inferences.

2.1.2 Defects in Software Models

Software defects may be related to an inappropriate comprehension of the information within the models. Granda et al. [14] present a classification for defects that are commonly found in UML models, such as:

Omission - The required information has been omitted.

Incorrect Fact - Some information in the model contradicts the list of requirements or general knowledge of the system domain.

Inconsistency - Information in one part of the model is inconsistent with information in other parts in the model.

Ambiguity - The information in the model is ambiguous. This can lead to different interpretations of information.

Extraneous Information - The information that is provided is not required in the model.

Redundant - Information is repeated in the model.

Inspection is a method used to identify defects with lower cost during the development process. According to Qazi et al. [16], the main purpose of an inspection is to identify defects to reduce costs and improve software quality. Defects in models can be associated to different information representation dimensions, these being Syntactic (relationship between model and the modeling language), Semantic (relationship of the model with the problem domain) and Pragmatic (relationship of the model with the stakeholders) [15].

2.2 Verification Items Base For The ComD2 Family

Using Grice's Cooperative Principle [13], ComD2 family uses the four maxims. We created verification items to support the identification of discrepancies (these discrepancies may be defects or not) based on these four maxims:

- Based on the maxim of **Quality**, we developed verification items for the identification of false information in the models, e.g. for the class diagram, we have: *"Do the classes have content that affects the quality of the model?".*
- Based on the maxim of **Quantity**, we developed verification items for the necessary content (and no more than necessary), in the models e.g., for the class diagram: *"Are all necessary classes of the problem domain in the diagram?"*.
- Based on the maxim of **Relation**, we developed verification items for the identification of information that is not relevant to the models, e.g.: *"Are classes relevant to system modeling?"*.
- Based on the maxim of **Manner**, we developed verification items for the identification of information that is not clear in the model, e.g.,: *"Are there classes and relationships with descriptions that are not clear?"*.

We observed that when the maxims are not respected in the models, they could cause defects in software. This occurs due to the lack of understanding of the consumers on the intention of the producers regarding the model. Thus, for each verification item, we used the defect classification that is presented by Granda et al. [14] (see Table 1). From the verification items, it is possible to classify the defects. For instance, for the verification item that is based on the maxim of Quantity, we relate this item as follows: *Are all necessary classes of the problem domain are in the diagram? If not, this may be an Omission discrepancy.* The verification items are divided into categories corresponding to the Grice's maxims, such as:

2.2.1 *Does the information in the model contain statements that are not true?*

- Are the elements being used inappropriately with the modeling language? If positive, this may be an Incorrect Fact discrepancy (Syntactic).
- Does the model have the true concepts for problem domain/solution domain? If not, this may be an Extraneous Information discrepancy (Semantic).
- Are there concepts implicit in the model that affect the quality of the information? If positive, this may be an Ambiguity discrepancy (Pragmatic).

2.2.2 Is the necessary information, and no more than necessary, present in the model?

- Are the elements used in a necessary way? If not, this may be an Omission discrepancy (Syntactic).
- Do the elements represent all the concepts involved in the problem domain/solution domain? If not, this may be an Omission discrepancy (Semantic).
- Do the elements that represent the concepts involved in the domain of the problem/domain of the solution have contents beyond what is necessary? If positive, this could be an Ambiguity and/or Redundancy discrepancies (Semantic and Pragmatic).

2.2.3 Is the information relevant to system modeling?

- Do the elements used in the model represent a logical sequence for the modeling language? If so, this may be an Incorrect Fact discrepancy (Synthetic).
- Are there concepts that are not relevant to the problem domain/solution domain? If so, this may be an Extraneous Information and/or Inconsistency discrepancies (Semantic and Pragmatic).

2.2.4 *Is the information difficult to understand in the model?*

- Are the elements being used in an organized way? If not, this may cause an Ambiguity discrepancy (Syntactic).
- Do the elements that represent all the concepts involved in the domain of the problem domain/solution domain are easy to understand? If not, this may be an Ambiguity discrepancy (Semantic).
- Do the elements that represent all the concepts involved in the domain of the problem domain/solution domain have contents that cause multiple interpretations? If so, this may be an Ambiguity and/or Redundancy discrepancies (Semantic and Pragmatic).

From this theoretical basis, it is possible to develop specific techniques for other software models. At this initial state, we have developed specific techniques for inspecting UML models, such as class diagrams, activities diagrams, and state machine diagrams, the three models that are commonly used in software development [9]. The next sections present the specific techniques for class diagrams, activities diagrams, and state machine diagrams.

3. ComD2 – Specific Technique for Class Diagrams

Does the information in the model contain statements that are not true?	
All elements	Are elements used improperly in the modeling language? If so, this may be an Incorrect Fact discrepancy (Syntactic). Are there concepts implicit in the diagram that affect the quality of the information? If positive, this may be an Ambiguity discrepancy (Pragmatic).
Classes	Do the classes in the diagram have concepts that are not involved in the problem domain? If so, this may be an Extraneous Information discrepancy (Semantic).
Attributes and Methods	Does attributes and methods have the true concepts for problem domain? If not, this may be an Extraneous Information discrepancy (Semantic).
Relationships	Is the cardinality of multiplicities correct in every relationship? If not, this may be an Inconsistence discrepancy (Semantic).
Is the necessary information, and no more than necessary, present in the model?	
All elements	Are all the necessary elements being used? If not, this may be an Omission (Syntactic) discrepancy.
Classes, Attributes and Methods	• Do the classes represent all the necessary concepts in the problem domain? If not, this may be an Omission discrepancy (Semantic). ***** Do the classes have unnecessary concepts? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic). • Do the Attributes and Methods necessary were represented? If not, this may be an Omission discrepancy (Semantic). ***** Do the Attributes and Methods have unnecessary concepts? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic). • According the problem domain, all *Association classes* necessary were established? If not, this may be an Omission discrepancy (Semantic). * There are *Association classes* unnecessary? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
Relationships	According to the problem domain, <u>all Association Relationships necessary</u> among classes were established? If not, this may be an Omission (Semantic) discrepancy. * There are <u>Association Relationships</u> unnecessary? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
	According to the problem domain, <u>all Aggregation Relationships necessary</u> among classes were established? If not, this may be an Omission (Semantic) discrepancy. * There are <u>Aggregation Relationships</u> unnecessary? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
	According to the problem domain, <u>all Composition Relationships necessary</u> among classes were established? If not, this may be an Omission (Semantic) discrepancy. * There are <u>Composition Relationships</u> unnecessary? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
	According to the problem domain, <u>all Generalization Relationships necessary</u> among classes were established? If not, this may be an Omission (Semantic) discrepancy. * There are <u>Generalization Relationships</u> unnecessary? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic
Is the information relevant to system modeling?	
All elements	Do the elements used represent a logical sequence for the class diagram? If so, this may be an Incorrect Fact discrepancy (Syntactic). Do the class diagram have low cohesion (responsibilities of one class are contained in another class)? If so, this may be an Extraneous Information and/or Inconsistency discrepancies (Semantic and Pragmatic).
Is the information difficult to understand in the model?	
All elements	• Are the elements disorganized? If so, this makes will it difficult to understand the information in the diagram, causing Ambiguity and Inconsistency discrepancies (Pragmatic). • Is it easy to understand the modeling solution in relation to the system? If positive, there is probably Ambiguity discrepancy (Semantic). • Do the elements have descriptions that multiple interpretations? If positive, there are probably Ambiguity and/or Redundancy discrepancies (Pragmatic). • Are there classes with similar names? If so, this may be an Ambiguity discrepancy (Pragmatic). • Are there classes with names that are difficult to understand? If so, this may be an Ambiguity discrepancy (Pragmatic). • Are there methods with names that are difficult to understand? If so, this may be an Ambiguity discrepancy (Pragmatic). • Are there elements with descriptions that are not clear? If so, this may be an Ambiguity discrepancy (Pragmatic).

4. ComD2 – Specific Technique for Activity Diagrams

Does the information in the model contain statements that are not true?	
All elements	Are elements used improperly in the modeling language? If so, this may be an Incorrect Fact discrepancy (Syntactic). Are there concepts implicit in the diagram that affect the quality of the information? If positive, this may be an Ambiguity discrepancy (Pragmatic).
Initial and Final Node	Are the initial and final node related to the incorrect activities? If so, this may be an Extraneous Information discrepancy (Semantic).
Activities	Do the activities represent different operations regarding the problem domain? If so, this may be an Extraneous Information discrepancy (Semantic).
Transition	Are there incorrect transitions regarding the problem domain? If so, this may be an Extraneous Information discrepancy (Semantic).
Decision	Are there incorrect decisions regarding the problem domain? If so, this may be an Extraneous Information discrepancy (Semantic).
Merge Node	Are there incorrect merge node(s) regarding the problem domain? If so, this may be an Extraneous Information discrepancy (Semantic).
Fork	Are there incorrect fork regarding the problem domain? If so, this may be an Extraneous Information discrepancy (Semantic).
Activity Partition	Are there incorrect activity partition regarding the problem domain? If so, this may be an Extraneous Information discrepancy (Semantic).
Is the necessary information, and no more than necessary, present in the model?	
All elements	Are all the necessary elements being used? If not, this may be an Omission (Syntactic) discrepancy.
Final Node	Do the final node unnecessary? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
Activities	Do the activities represent all the necessary concepts in the problem domain? If not, this may be an Omission discrepancy (Semantic). * Do the activities unnecessary were represented? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
Transition	Do the transitions represent all the necessary concepts in the problem domain? If not, this may be an Omission discrepancy (Semantic). * Do the transitions unnecessary were represented? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
Decision	Do the decisions represent all the necessary concepts in the problem domain? If not, this may be an Omission discrepancy (Semantic). * Do the decisions unnecessary were represented? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
Merge Node	Do the merge nodes unnecessary were represented? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
Fork	Do the fork unnecessary were represented? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
Activity Partition	Do the activities partitions unnecessary were represented? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
Is the information relevant to system modeling?	
All elements	Do the elements used represent a logical sequence for the activity diagram? If so, this may be an Incorrect Fact discrepancy (Syntactic).
Is the information difficult to understand in the model?	
All elements	• Are the elements disorganized? If so, this makes will it difficult to understand the information in the diagram, causing Ambiguity and Inconsistency discrepancies (Pragmatic). • Is it easy to understand the modeling solution in relation to the system? If positive, there is probably Ambiguity discrepancy (Semantic). • Do the elements have descriptions that multiple interpretations? If positive, there are probably Ambiguity and/or Redundancy discrepancies (Pragmatic).

5. ComD2 – Specific Technique for State Machine Diagrams

Does the information in the model contain statements that are not true?	
All elements	Are elements used improperly in the modeling language? If so, this may be an Incorrect Fact discrepancy (Syntactic). Are there concepts implicit in the diagram that affect the quality of the information? If positive, this may be an Ambiguity discrepancy (Pragmatic).
Initial and Final State	Are the initial and final state related to the incorrect states? If so, this may be an Extraneous Information discrepancy (Semantic).
State	Are there states regarding the problem domain? If so, this may be an Extraneous Information discrepancy (Semantic).
Event and Transition	Do the event/transition description have information different from the class/object being modeled? If so, this may be an Extraneous Information discrepancy (Semantic).
Is the necessary information, and no more than necessary, present in the model?	
All elements	Are all the necessary elements being used? If not, this may be an Omission (Syntactic) discrepancy.
Event and Transition	Do the event/transition represent all the necessary concepts in the problem domain? If not, this may be an Omission discrepancy (Semantic). * Do the event/transition necessary were represented? If so, this may be an Ambiguity and/or Redundancy discrepancies (Pragmatic and Semantic).
Is the information relevant to system modeling?	
All elements	Do the elements used represent a logical sequence for the state machine diagram? If so, this may be an Incorrect Fact discrepancy (Syntactic).
Is the information difficult to understand in the model?	
All elements	• Are the elements disorganized? If so, this makes will it difficult to understand the information in the diagram, causing Ambiguity and Inconsistency discrepancies (Pragmatic). • Is it easy to understand the modeling solution in relation to the system? If positive, there is probably Ambiguity discrepancy (Semantic). • Do the elements have descriptions that multiple interpretations? If positive, there are probably Ambiguity and/or Redundancy discrepancies (Pragmatic).

6. ARTIFACTS TO SUPPORT THE INSPECTION

In this section, we present the artifacts that can be used to support the inspection of the software models with ComD2 techniques. These artifacts are described below:

6.1 Inspection Form

Name: _____ **Initial Time: _____End Time:_____**

Defect Type	Dimension(s) of model Representation	Defect Description

6.2 Questionnaire for Evaluation of ComD2 Techniques

Name: _____

Please, answer the following questions regarding your experience with the ComD2 technique(s):

1. What is your perception with the use of the technique(s)?.

2. Do the information representation dimensions help to understand the defects that could undermine the understanding of the model?

3. What are the difficulties with using the technique(s)?

REFERENCES

[1] A. H. Reed and L.V. Knight, "Effect of a virtual project team environment on communication-related project risk", International Journal of Project Management, vol. 28 (5), 2010, pp. 422–427.

[2] E. Diel, S. Marczak, D. S. Cruzes, "Communication Challenges and Strategies in Distributed DevOps", Proceedings of the 11th International Conference on Global Software Engineering (ICGSE 2016), 2016, pp. 24-28.

[3] V. Käfer, "Summarizing software engineering communication artifacts from different sources", Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017), 2017, pp. 1038-1041.

[4] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, J. Still, "The impact of agile practices on communication in software development", Empirical Software Engineering, vol. 13 (3), 2008, pp. 303-337.

[5] P. Ralph, M. Chiasson and H. Kelley, "Social theory for software engineering research", Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE '16), 2016, pp. 44-55.

[6] R. M. de Mello, E. N. Teixeira, M. Schots, C. M. L. Werner and G. H. Travassos, "Verification of software product line artefacts: a checklist to support feature model inspections", Journal of Universal Computer Science, vol. 20(5), 2014, pp. 720-745.

[7] A. M. Qazi, S. Shahzadi and M. Humayun, "A comparative study of software inspection techniques for quality perspective", International Journal of Modern Education and Computer Science, vol. 8 (10), 2016, pp. 9-16.

[8] J. M. Brown, G. Lindgaard, and R. Biddle, "Collaborative events and shared artefacts: Agile interaction designers and developers working toward common aims," Proceedings - 2011 Agile Conference, Agile 2011, pp. 87–96.

[9] G. Reggio, M. Leotta, F. Ricca, and D. Clerissi, "What are the used activity diagram constructs? A survey", Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2014), 2014, pp. 87–98.

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

[11] C. S. De Souza, The Semiotic Engineering of Human-Computer Interaction (Acting with Technology). The MIT Press, 2005.

[12] Clarisse Sieckenius de Souza, Renato Fontoura de Gusmão Cerqueira, Luiz Marques Afonso, Rafael Rossi de Mello Brandão and Juliana Soares Jansen Ferreira. 2016. Software Developers as Users: Semiotic Investigations in Human-Centered Software Development. In Springer International Publishing Switzerland. DOI 10.1007/978-3-319-42831-4.

[13] H. P. Grice, "Logic and conversation". Syntax and Semantics 3: Speech arts, ed. Peter Cole and Jerry Morgan, 1975, pp. 41–58.

[14] M. F Granda, N. Condori-fernández, T. E. J. Vos, O. Pastor, "What do we know about the defect types detected in conceptual models?", Proceedings of the IEEE 9th Int. Conference on Research Challenges in Information Science (RCIS 2015), 2015, pp. 96–107.

[15] M. Priyanka and R. Phalnikar, "Generating UML diagrams from natural language specifications", International Journal of Applied Information Systems, vol. 1(8), 2012, pp. 19-23.

[16] A. M. Qazi, S. Shahzadi and M. A Humayun, "Comparative study of software inspection techniques for quality perspective", International Journal of Modern Education and Computer Science, vol. 10 (8), 2016, pp. 9-16, DOI: 10.5815/ijmecs.2016.10.02.

[17] G. Travassos, F. Shull, M. Fredericks, V. Basili, "Detecting defects in object-oriented designs: using reading techniques to increase software quality", Proceedings of XIV ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, And Applications, 1999, pp. 47-56.