Technique for Inspecting MoLIC Interaction Diagrams

Adriana Lopes[1], Natasha Valentim[1], Bruna Moraes[1], Renata Zilse[2], Tayana Conte[1]

[1] USES – Grupo de Usabilidade e Engenharia de Software
PPGI – Programa de Pós-Graduação em Informática Instituto de Computação
Universidade Federal do Amazonas (UFAM) Manaus, AM – Brazil

{adriana, natashavalentim, bmf, tayana}@ufam.edu.br

[2] Samsung Research Brazil
Av. Cambacica, 1200 - Building 1, Parque Resedás - Campinas – SP, Brasil

renata.borges@samsung.com

USES Technical Report
RT-USES-2017-0008
April, 2017

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

Technique for Inspecting MoLIC Interaction Diagrams

Adriana Lopes[1], Natasha Valentim[1], Bruna Moraes[1], Renata Zilse[2], Tayana Conte[1]

[1] Instituto de Computação – Universidade Federal do Amazonas (UFAM)
Av. General Rodrigo Octávio, 6200, Coroado I – Manaus – AM – Brasil

[2] Samsung Research Brazil

{adriana, natashavalentim, bmf, tayana}@ufam.edu.br,
renata.borges@samsung.com

Abstract

Techniques that help in understanding user needs are increasingly being used in Software Engineering to improve the acceptance of applications. In the design stage, interaction models were employed to develop other artifacts, such as prototypes. We present a technique developed for the inspection of interaction models. This technique reduced the spread of defects in the interaction models. The technique was important to apply the inspection of interaction models before they are used as basis for the development of other artifacts.

A Technique for Inspecting MoLIC Interaction Diagrams

Software inspection is a method that assists in the verification of artifacts constructed during the development of a system with a rigorous and well defined process [Fagan, 1976] [Taba and Siew, 2012]. The main purpose of an inspection is to identify defects in software development to reduce costs and improve software quality [Qazi et al., 2016]. The importance of conducting inspections during the development process is broadly cited in the literature [Aurum et al., 2002] [Misra et al., 2014].

We developed a technique for Inspecting MoLIC Interaction Diagrams (IMID) based on the different types of defects found in the MoLIC diagram for the HCDP application from Sprint 3 [de Mello et al. , 2014]. Among the types of defects, we can cite: Omission, Ambiguity, Incorrect Fact, Inconsistency and Extraneous Information. The IMID verification items only evaluate the consistency of the MoLIC diagrams with the software requirements. We believe that these verification items can be instantiated for other interaction modeling notations, as long as the elements have the same purpose in the interaction modeling. Below we present the verification items (Verification Items for the MoLIC Element *vs* Software Requirements) related to the MoLIC elements (#E). In addition, we provide the support form for that other practitioners can use them in their evaluation of interaction models

Technique for Inspecting MoLIC Interaction Diagrams (IMID)

E#	Verification Items for the MoLIC Element vs Software Requirements
Opening point **Closing point**	**Verification item 1:** Was the element used in the model to represent the start / end of the interaction? If not, report it as an Omission defect. (***Generated problem****: This defect may impair understanding the beginning of the interaction by the team members.*) **Verification item 2:** If the previous item occurs, does the element represent the beginning / end of the interaction according to the requirements? If not, report it as an Inconsistency defect. (***Generated problem****: This defect may impair the consistency of requirements with the appropriate start of the user interaction in the system*).
Scene, Signs, Dialogues and Structures of dialogues	**Verification item 3:** Are all requirements represented in the interaction model? If not, report it as an Omission defect. (***Generated problem:*** *This defect may hinder the development of a necessary requirement*.) **Verification item 4:** Are there inconsistent requirements in the interaction model? If so, report it as an Inconsistency type defect. (***Generated problem****: This defect may make the software inconsistent with the requirements*.) **Verification item 5:** Is there information in the interaction model that is not in the context of the requirements? If so, report it as an Extraneous Information defect. (***Generated problem****: This defect may make the software present irrelevant information*.) **Verification item 6:** Is it possible to understand all the information about the requirements in the interaction model in a clear way? If so, report it as an Ambiguity defect. (***Generated problem****: This defect may cause the insertion of other defects in the developed artifacts, since each team member may have a different interpretation.*) **Verification item 7:** Have all requirements been correctly represented in the interaction model? If not, report it as an Incorrect Fact defect. (***Generated problem****: This defect may hinder the development of correct requirement.)*
Transition Utterance **Breakdown recovery utterance**	**Verification item 8**: Are there any omissions on interactions required for the software features? If so, report it as an Omission defect. (***Generated problem****: This defect may impair the user interaction with the system*.) **Verification item 9**: Is the content of the interactions between the features consistent with the requirements? If not, report it as an Inconsistency defect. (***Generated problem****: This defect may make the user interaction inconsistent from the requirements point of view*.) **Verification item 10**: Is the content of the interactions between the functionalities in the context of the requirements? If not, report it as an Extraneous Information defect. (***Generated problem****: This defect can also impair the user interaction from the point of view of the requirements*.) **Verification item 11**: Does the content of the interactions between the features provide multiple interpretations? If so, report it as an Ambiguity defect. (***Generated problem****: This defect may provide for the insertion of other defects in the developed artifacts, since each team member may have a different interpretation*.) **Verification item 12**: Have all interactions been correctly represented in the interaction model? If not, report it as an Incorrect Fact defect. (***Generated problem****: This defect may hinder the development of correct interactions*.)
System process	**Verification item 13:** Was the element required for the interpretation of an applied user action? If not, report it as an Omission defect. (***Generated problem***: *This defect can impair proper user interaction in the system after a certain action, such as an interpretation for logging into the system*) **Verification item 14:** Was correct feedback used after the system processing? If not, report it as an Omission defect. (***Generated problem***: *This defect may impair the proper user interaction in the system, such as the result of attempting to logging into the system*.)
Ubiquitous access	**Verification item 15:** Can the features associated with this element be accessed at any time in the user-system interaction? If not, report it as an Inconsistency defect. (***Generated problem****: This defect may make the user interaction inconsistent from the requirements point of view.)*

Inspection Form for Interaction Models

Name: _____ Initial Time_____ End Time: _____

Number Defect	Verification Item	Defect Type	Defect Description

References

Aurum , A., Petersson, H., Wohlin, C. (2002) State-of-the-Art: Software Inspection after 25 years, In: Journal of Software, Testing, Verification and Reliability, 12(3): 133-154.

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

Fagan, M. E. (1976) Design and code inspections to reduce errors in program development. IBM Journal of Research and Development 15 (3): 182-211.

Misra, S., Fernández, L., Colomo-Palacios, R. (2014) A simplified model for software inspection. Journal of Software: Evolution and Process, 26(12): 1297–1315.

Qazi, A. M., Shahzadi, S., Humayun, M. (2016) A Comparative Study of Software Inspection Techniques for Quality Perspective. International Journal of Modern Education and Computer Science, 10 (1): 9-16.

Taba, N. H., Siew H. O. (2012) A Scenario Based Model to Improve the Quality of Software Inspection Process. In: Proceedings of the 4th International Conference on Computational Intelligence, Modelling and Simulation (CIMSIM '12), 194-198.