Validation Based on Observation of the Behavior of Distributed Components

Tatiane Macedo Prudencio Lopes
ITA – Aeronautical Institute of Technology
tatiane@ita.br

Clovis Torres Fernandes
ITA – Aeronautical Institute of Technology
clovistf@uol.com.br

Abstract

In this paper, we present the results of a research in which the validation activity of distributed software components is accomplished via formal models based on the observation of the component behavior. The proposal is to present an improved manner to validate the component by using its real behavior, and not only insufficient documentation and suppositions.

1. Research topic and Motivation

Validation is essential in software development because it is responsible for the correspondence between the functionalities required by the client and those specified and implemented by the developer. Validation corresponds to the application of several test levels, such as unit, integration, system, acceptance, regression, etc., based on the test scenario created during the verification activity [1][2]. Once there is a difference between test results and specified results, it is an indication that the components were not implemented according to the specifications.

In component-based software development, validation is not a trivial task, for system quality is dependent upon individual component quality. In this context, the difficulty to accomplish the validation tests comes from the fact that the components may be created by different suppliers, which use distinct techniques, methods and tools in the production process.

This miscellany which occurs in software validation becomes even more challenging when the software is distributed among different teams, located, e.g. in different countries. As an example, one specific component may be developed and tested by a company in Brazil which needs the contribution of a component developed and tested in China, which in turn, may need a component developed and tested in France. How can these clients and suppliers communicate with each other efficiently once they are endowed with different cultures and languages?

Each company has its own techniques, methods and tools according to its technical skills related to tests, its culture, its language, etc. Information related to component tests, necessary for the test execution or resulting from executed tests, even when available to other clients, will be difficult to understand once they do not feature a sharing pattern. Moreover, the internal structure of outsourced components is generally unknown and the documentation is not sufficient to work with all of the details and their interaction with other components.

A solution to this problem may be found by using a common validation process among all of the teams which test each component. This process is based on the observation of the behavior of each component. This behavior is modeled in such a manner that client teams are able to test its real behavior without the need to make suppositions or jump to conclusions about it.

In Figure 1 is showed the common validation process used by different teams that is based on component behavior.

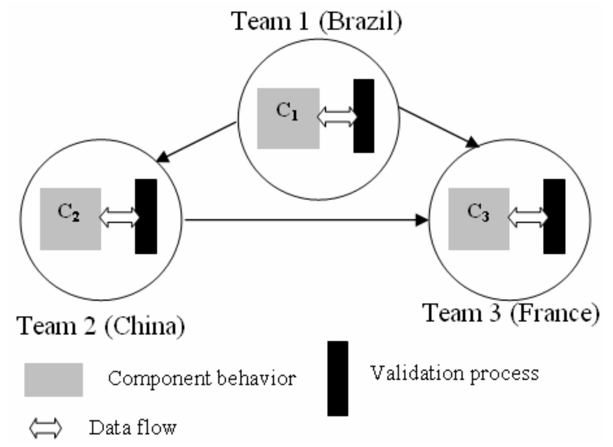

Figure 1. Example of teams that use a common validation process.

2. Proposed Technique

The internal structure of a component is generally unknown and the documentation is not sufficient to solve details about its interaction with other components.

In order to identify the relevant tests of the interactions, these tests are based on formal models of the components. However, the COTS are not often provided with complete and precise models which would help the integration tests. In practice, maintaining formal models is unrealistic because COTS evolve along time, which quickly invalidates the original project. A solution to this problem may be found by generating models directly from the components.

In this case, the component system itself will act as an oracle during the tests. To accomplish this, an iteration technique is created to learn formal models from black-box components.

In Figure 2, the steps of the algorithm for the validation of the distributed component from the observation of its behavior are depicted, based on [3] and [4].

1. In the first step, each test team E_1, E_2, E_3, ..., E_N defines an input alphabet I_C for its component C. This alphabet corresponds to the contribution that the component needs from others as well as its own independent inputs. All relevant parameter values for each team E_i are considered.
2. Each component from each team is tested separately using a learning algorithm [4] until balanced, complete and consistent observation tables are determined. The output alphabet is determined along with the output parameters. When the procedure is accomplished, we have the first model $C^{(1)}$ for C. This procedure is referred to as Unit Testing.
3. Components are integrated, i.e. some of the outputs of a certain component will be used as inputs to the connected interface of another component. This integration among components is tested in two stages.
4. In a first stage, the expected scenarios from the integration correspondent to the system as a whole are tested. In this stage, input parameter values are selected according to the test scenario to build parameterized input sequences, observe parameterized output sequences and determine when a test scenario is respected. Scenarios act as *oracles*. Faults can be detected, or a discrepancy within the inferred model can be identified leading to an incremental refinement of the model.
5. In a second stage, tests are generated from the component models. The real observed outputs (internal and from the environment) are recorded and compared to the outputs generated by the models. Tests are carried out until a discrepancy is found in one of the model's previsions or some covering criterion in the model is reached. The categorization of discrepancies as faults may require input from the specialist.
6. When the covering criterion is reached, the domain specialist checks upon the component models and tests the results. In cases where not expected behavior is identified in the component, it may be replaced, and unit and integration tests are iterated.
7. In both stages, discrepancies may lead to model refinement. The counterexamples found are inserted back into the learning algorithm to extend the models and the stage is iterated with new $C^{(i+1)}$ components.
8. The process ends after the domain specialists approve the results.

Figure 2. Steps of distributed component validation.

3. Contributions

The main contributions of this work are:

- The validation process is the same for all teams, i.e. all test artifacts generated or needed by one team must be standardized.
- The test tools, methods and techniques used by all the teams are the same.
- There is an efficient exchange of information between the component supplier and client teams.
- There is an intra-component (within the own team) and inter-component (among teams) dependency order which must be followed by the execution of the test cases. The relation of intra-component dependency can be found in [6] and [7]. The relation of inter-component dependency is presented in [8].
- There is mutual cooperation among all test teams.

5. References

[1] R. V. Binder (2000). Testing Object-Oriented Systems. United Satates: Addison-Wesley.

[2] M. J. Harrold, A. Orso, D. Rosenblum, G. Rothermel, M. L. Soffa, H. Do (2001). Using component metadata to support the regression testing of component-based software. In: Proceedings of the International Conference on Software Maintenance, pp. 716-725. IEEE Computer Society Press.

[3] M. Shahbaz (2006). Incremental Inference of Black-Box Components to Support Integration Testing. In: Proceedings of the Testing: Academic & industrial Conference on Practice and Research Techniques. TAIC-PART. IEEE Computer Society, Washington, DC, pp. 71-74.

[4] M. Shahbaz, K. Li, R. Groz (2007). Learning Parameterized State Machine Model for Integration Testing. In: Proceedings of the 31st Annual international Computer Software and Applications Conference - Vol. 2- (COMPSAC 2007). COMPSAC. IEEE Computer Society, Washington, DC, pp. 755-760.

[6] T. M. P. Lopes (2004). Uma Estrutura para o Plano de Codificação e Testes no Desenvolvimento Evolutivo de Software Orientado a Objetos. Dissertação de Mestrado, ITA, São José dos Campos – São Paulo.

[7] T. M. P. Lopes, C. T. Fernandes (2007a). Planejamento Integrado das Atividades de Codificação e Testes na Orientação a Objetos no Nível de Granularidade dos Métodos da Classe. In: X Workshop Iberoamericano de Ingenieria de Requisitos Y Ambientes de Software – IDEAS 2007.

[8] T. M. P. Lopes, C. T. Fernandes (2007b). Management and Control of the Coding and Testing Activities of Component-based Software. In: 31st Annual IEEE International Computer Software and Applications Conference (COMPSAC). IEEE Computer Society Press.