Extending the SOLIMVA methodology to address incompleteness in software specifications

Valdivino Alexandre de Santiago Júnior[1], **Nandamudi Lankalapalli Vijaykumar**[2]

[1]Programa de Doutorado em Computação Aplicada – CAP
Instituto Nacional de Pesquisas Espaciais – INPE

[2]Laboratório Associado de Computação e Matemática Aplicada – LAC
Instituto Nacional de Pesquisas Espaciais – INPE

valdivino@das.inpe.br,vijay@lac.inpe.br

Abstract. *Incompleteness in software requirements specifications affects the next software artifacts, including source code, developed within the software development lifecycle. This work extends the SOLIMVA methodology to address incompleteness in software specifications. We used Model Checking combined with k-permutations of n values of variables and specification patterns to tackle this problem. We present the results of applying our approach to a software product in the space application domain.*

Keywords: *SOLIMVA Methodology, Incompleteness, Software Specifications, Model Checking, k-permutations of n.*

1. Introduction

There are several publications in the literature that prove that problems in requirements are serious factors that affect the quality of software products. In a survey over 8,000 projects undertaken by 350 US companies, managers identified requirements as the main cause of project failures in accordance with the following problems: the lack of user involvement (13%), requirements incompleteness (12%), changing requirements (11%), unrealistic expectations (6%), and unclear objectives (5%) [van Lamsweerde 2000].

Incompleteness, inconsistency, and, especially in Natural Language (NL) requirements specifications, ambiguity are among the types of defects found in software requirements specifications. Since software requirements specifications are created early within the software development lifecycle, their defects affect the next software artifacts, including source code, to be developed.

In a previous work we presented SOLIMVA, a methodology aiming at model-based test case generation considering NL requirements deliverables [Santiago Júnior and Vijaykumar 2011]. In this work, we extend the SOLIMVA methodology to address incompleteness in software specifications (software requirements specifications, communication protocol specifications, ...). We used Model Checking [Baier and Katoen 2008] combined with k-permutations of n values of variables and specification patterns [Dwyer et al. 1999] to tackle this problem. We applied this extension of SOLIMVA to a significant software product in the space application domain. We detected 21 incompleteness defects ranging in severity from Low to High, and with the impact's attributes Installability, Documentation, and Usability of the Orthogonal Defect Classification (ODC).

2. A proposal to detect incompleteness in software specifications

The extension of the SOLIMVA methodology to address incompleteness in software specifications is peformed by two activities: *Analyze Incompleteness* and *Improve Specifications*. The most important activity to deal with the problem of incompleteness is *Analyze Incompleteness* which will be described in the sequence.

Our proposal to identify incompleteness in software specifications uses Model Checking, specification patterns and 2-permutations of n values of characteristics of specifications. However, there are important changes of philosophy which differ the way Model Checking is applied within the *Analyze Incompleteness* activity. As our goal is to detect defects in software specifications, software specifications are used not to develop properties but rather to develop the model of the system. However, it is necessary to generate the properties preferably in a way independent of the requirements in the software specifications. We achieved this by combining specification patterns with 2-permutations of n values of characteristics. This is discussed below.

First, it must be chosen a primary characteristic ($prim$) and its values ($valprim_i, 1 \leq i \leq l$). The Verification and Validation (V&V) Team member must then select some $valprim_i$ in order to generate the model. For each selected $valprim_i$, secondary characteristics ($sec_j, 1 \leq j \leq m$) and their values ($valsec_{jk}, 1 \leq k \leq n$) must be obtained. Then, a finite-state model for the selected $valprim_i$ must be generated and also simulated to determine and correct possible defects which arise by translating the software specifications into the model. Primary and secondary characteristics are attributes of software specifications.

Specification patterns come into picture to formalize properties in Computation Tree Logic (CTL) for each $valsec_{jk}$ and according to the **Absence Pattern and Globally Scope** ($propertyAbs_{jk}$). Each property is generated as follows: $\forall\Box\neg(prim = valprim_i \wedge sec_j = valsec_{jk})$. Thus, if the model does not satisfy this property this means that $valsec_{jk}$ is present in the software specifications regarding $valprim_i$; otherwise, if the property is satisfied then it is not present. But, the V&V Team member must predict in advance whether the property must or must not be satisfied. In other words, if $valsec_{jk}$ must appear in the software specifications then the expected result must be $false$. On the other hand, if $valsec_{jk}$ must not appear, the expected result must be $true$. An incompleteness defect is detected if there is a discrepancy between the satisfaction or non-satisfaction of the property by the model and the expected result.

After that, some sec_j must be selected for 2-permutations of n analysis and, for each chosen sec_j, clusters ($cluster_p$) of selected $valsec_{jk}$ are determined. Application of 2-permutations of n is then accomplished for each $cluster_p$, and each permutation $t = \{t1, t2\}$ will derive two types of CTL properties. The first property is based on the **Soft Response Pattern and Globally Scope** ($propertySoftResp_t$) and is as follows: $\forall\Box((prim = valprim_i \wedge sec_j = valsec_{t1}) \rightarrow \exists\Diamond(prim = valprim_i \wedge sec_j = valsec_{t2}))$. The second property is according to the **Precedence Pattern and Globally Scope** ($propertyPrec_t$) and is as follows: $\neg\exists[\neg(prim = valprim_i \wedge sec_j = valsec_{t1}) \cup ((prim = valprim_i \wedge sec_j = valsec_{t2}) \wedge \neg(prim = valprim_i \wedge sec_j = valsec_{t1}))]$.

After all properties are formalized, Model Checking is applied. If $propertySoftResp_t$ is not satisfied then an incompleteness defected is detected. If $valsec_{t1}$ occurs in the

software specifications then the Soft Response Pattern and Globally Scope requires that there is some path in which $valsec_{t2}$ occurs following $valsec_{t1}$. The interpretation is that if this property does not hold then the software specifications were not complete enough to predict such behavior, i.e. $valsec_{t2}$ occurs as a response to $valsec_{t1}$. Hence an incompleteness defect is detected.

Being $propertySoftResp_t$ satisfied then the result depends on $propertyPrec_t$. If $propertyPrec_t$ is also satisfied then an incompleteness defect is also detected. The reasoning behind this is that if $propertyPrec_t$ is satisfied it is because $valsec_{t1}$ always occurs before (precede) $valsec_{t2}$ and this shows that the software specifications were not complete enough to predict the opposite, namely, $valsec_{t2}$ occurs before $valsec_{t1}$. However, the non-satisfaction of $propertyPrec_t$ indicates that no incompleteness defect exists.

3. Case study: space application software product

We applied the extension of the SOLIMVA methodology to address incompleteness in software specifications to a space application software product, Software for the Payload Data Handling Computer (SWPDC) [Santiago Júnior and Vijaykumar 2011]. SWPDC is embedded into the Payload Data Handling Computer (PDC) which communicates with the On-Board Data Handling (OBDH) Computer, the satellite platform computer.

We analyzed two deliverables to detect incompleteness: SWPDC's Software Requirements Specification and PDC-OBDH Communication Protocol Specification. The primary characteristic is the PDC's operation mode ($opmode$) which has four possible values, i.e. $valprim = \{initiation, safety, nominal, diagnosis\}$. We selected only the main operation mode of PDC, the Nominal Operation Mode ($valprim_3 = nominal$), for the analysis. Then, we selected four secondary characteristics, i.e. $sec = \{service, cmd, resp, action\}$ where: $service$ represents services supported by SWPDC; cmd are commands defined in the PDC-OBDH Communication Protocol Specification; $resp$ are responses defined in the PDC-OBDH Communication Protocol Specification; and $action$ represents some actions that SWPDC shall perform.

The model created for $valprim_3 = nominal$ has 71 reachable states. We used the NuSMV Model Checker. After generating and simulating the model, we created 70 CTL properties according to the Absence Pattern and Globally Scope. For instance, in order to verify whether the command TRANSMIT SCIENTIFIC DATA ($valsec_{24} = txSci$) is present in the software specifications, this CTL property is created: $\forall\Box\neg(opmode = nominal \wedge cmd = txSci)$. We repeated this template of CTL property for each $valsec_{jk}$ where $1 \leq j \leq 4$. By applying Model Checking, we detected 5 incompleteness defects.

In order to generate CTL properties in accordance with the Soft Response and Precedence Patterns we must choose some secondary characteristics. We selected the most relevant secondary characteristic: the commands that the OBDH can send to PDC ($sec_2 = cmd$). We then defined 5 clusters of selected $valsec_{2k}$ and applied 2-permutations of n considering each of these 5 clusters. Based on each permutation t we generated a pair of CTL properties: $propertySoftResp_t$/$propertyPrec_t$. For instance, for $cluster_1 = \{txSci, retAnsw, txSciEBuf, txSciinc\}$, 12 permutations were created due to 2-permutations of n: $\{txSci, retAnsw\}, \ldots, \{txSciinc, txSciEBuf\}$. Thus, pairs of

CTL properties were generated as follows:

$$\forall\Box((opmode = nominal \wedge cmd = txSci) \rightarrow \exists\Diamond(opmode = nominal \wedge cmd = retAnsw)),$$
$$\neg\exists[\neg(opmode = nominal \wedge cmd = txSci) \cup ((opmode = nominal \wedge cmd = retAnsw) \wedge \neg(opmode = nominal \wedge cmd = txSci))].$$

We generated 146 CTL properties according to the Soft Response Pattern and Globally Scope, and 146 CTL properties in accordance with the Precedence Pattern and Globally Scope. We detected 16 incompleteness defects due to the combination of these two patterns/pattern scopes. In total, we produced 362 CTL properties and detected 21 incompletenesses in accordance with the following distribution: (i) impact's attributes of ODC - Installability = 9; Documentation = 3; Usability = 9; (ii) defect's severity - Low = 16; Medium = 1; High = 4.

4. Conclusions

This paper extends SOLIMVA, a methodology aiming at model-based test case generation considering NL requirements deliverables, to address incompleteness in software specifications. We used a combination of three techniques to achieve this goal: Model Checking, k-permutations of n values of variables, and specification patterns.

We proposed an approach to generate properties independently of the requirements in the software specifications because the goal was to find defects (incompleteness) in the software specifications. Moreover, specification patterns are not used in the traditional manner where, based on a requirement, a pattern and the scope within the pattern that mostly characterize such requirement are identified and properties are then generated in Linear Temporal Logic (LTL), CTL. Instead, we identify characteristics and use combinatorics in conjunction with specification patterns to generate properties. In addition, a different interpretation of pattern/pattern scope is provided aiming at detecting incompleteness defects. Our proposal has demonstrated its feasibility by its application to a representative case study in the space domain and the detection of incompleteness defects.

References

Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. The MIT Press, Cambridge, MA, USA. 975 p.

Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. (1999). Patterns in property specifications for finite-state verification. In *Proceedings...*, pages 411–420, New York, NY, USA. International Conference on Software Engineering (ICSE), ACM.

Santiago Júnior, V. A. and Vijaykumar, N. L. (2011). Generating model-based test cases from natural language requirements for space application software. *Software Quality Journal*, pages 1–67. DOI: 10.1007/s11219-011-9155-6, URL: http://dx.doi.org/10.1007/s11219-011-9155-6.

van Lamsweerde, A. (2000). Requirements engineering in the year 00: a research perspective. In *Proceedings...*, pages 5–19, New York, NY, USA. International Conference on Software Engineering (ICSE), ACM.