27

Software Quality Improvement: Two Approaches to the Application of Formal Methods

A. Alapide, S. Candia, M. Cinnella, S. Quaranta
Space Software Italia
Viale del Lavoro, 101 -- 74100 Taranto -- Italy
Tel: +39 99 4701619 Fax: +39 99 4701777 E-mail: cinnella@ssi.it

Abstract

This paper illustrates two different approaches for the application of Formal Methods (FM): *integrated-parallel* and *after-the-fact*. In the first approach FMs have been applied integrated and in parallel with structured methods starting from the design phase. In the second approach FMs have been applied after the whole application code had already been developed, before the delivery, to derive an abstract specification of the s/w system and verify that the most critical properties hold.

Both approaches have been adopted in the development of a real application in the domain of the Air Traffic Control, whose purpose is to predict and detect potential air conflicts.

The results show that FMs can improve the quality of the software process and products. In particular the accuracy of the final documentation improves and the number of early discovered errors increases.

The paper provides general guidelines for the integration of formal and structured methods and presents the documentation outline which has been defined to comment the formal specifications, in the framework of the project applicable standards: 2167-A military standard and ESA PSS-05-0 and PSS-01-0.

Finally the paper makes an analysis of eight software quality factors, showing also the typology of the discovered errors with the two after-the-fact and integrated-parallel approaches with respect to the traditional development approaches. One conclusion is that FMs provide a real support in developing better quality software, identifying errors, which sometimes, with traditional approaches, remain undiscovered till and after the software delivery.

Keywords

Formal methods, RAISE, Ada/Teamwork, 2167-A standard, quality factors

1 INTRODUCTION

SSI (Space Software Italia) has applied the RAISE Formal Method (FM) to develop a software application in the domain of the Air Traffic Control for the Alenia Radar System Division. The overall aim in the application has been to innovate the software development

S. Bologna et al. (eds.), *Achieving Quality in Software*

process integrating FMs with structured methods, applying them to develop a wide part of a TCA CSC (Computer Software Component), the Detector, which was critical because in charge of detecting the potential conflicts.
This approach to FM introduction was smooth. It allowed to obtain some of the benefits arising from FM adoption -- both in the product and in the process -- without overspending with respect to the effort that would have been required using a more traditional approach. Moreover it allowed, to both SSI and the customer, to learn some lessons, which can constitute the basis for ensuring a future low risk transition towards a larger scale adoption of FMs.
The project focused essentially on a few, but well defined process and product quality factors. The four most relevant key *process* factors which were considered are:

- Early error discovery;
- Effective communication with the customer;
- Easiness of maintenance;
- Compliance with the applicable process standards.

The four most relevant *product* quality factors were:

- Correctness of the final software;
- Readibility, completeness and consistency of the final documentation;
- Compliance with the applicable product standards;
- Reusability of both code and documentation.

Several approaches for the integration of structured and formal methods [5], [8], [10] have been proposed. For the development of the Detector CSC two approaches based on the ones presented in [5] have been followed: *integrated-parallel* and *after-the-fact.* After their brief description, the article presents an analysis of the two approaches, focusing on on each of the eight above mentioned quality factors. Moreover it compares the results with the ones obtained with a traditional approach for the development of the other TCA application CSCs.

2 APPLICATION BACKGROUND

The developed application is *TCA (Traffic Conflicts Alert) Analyzer*, a software system in charge of elaborating radar data and predicting potential air conflicts belonging to the following classes:

- STCA (Short Term Conflicts Alert): conflicts among planes;
- MSAW (Minimum Safe Altitude Warning): planes going lower than the minimum safe altitude;
- DAIW (Dangerous Area Infringement Warning): planes entering a restricted area.

The CSC Detector is a critical component of TCA. The development process was based on the SSI SQS (Software Quality System) for projects having a medium criticality level. SSI SQS has been accredited by ESA (European Space Agency) for the PSS-05-0 and PSS-01 series and is compliant with the Nato and the 2167-A DoD standard (required by the customer).
The RAISE FM was selected for the application development because SSI had a previous related experience on it and because the CEC (Commission of the European Communities)

sponsored the application within the LaCoS (Large Scale Correct Systems) ESPRIT II (European Strategic Programme for the Information Technology) project.

3 RAISE AND TEAMWORK/ADA

3.1 RAISE

The RAISE (Rigorous Approach to Industrial Software Engineering) FM is based on the formal language RSL (RAISE Specification Language), the RAISE method and a powerful toolset.
The *RAISE Specification Language* (RSL) [1] is provided with structuring mechanisms that allow one to build modularized specifications of complex systems with layering. It includes constructs to model concurrency and allows several specification styles at different level of detail (from *abstract* to *concrete*).
The *RAISE method* [2] allows two types of formal proofs: *inter-level* proofs and *intra-level* proofs. The former deal with proving that the specification of level i+1 is consistent with the specification of level i (static and dynamic *development relations* in fig. 1), while the latter deals with proving that the specification of level i is consistent and satisfies the stated critical requirements.
The *RAISE toolset* [3] allows to edit RSL specifications with automatic correctness checks, supports the automatic generation of confidence conditions and the automatic verification of the static development relation. It provides support to prove the dynamic relation and allows to edit and prove theories, confidence conditions and the dynamic development relations. Moreover it allows to translate automatically in Ada or C++ the RSL concrete specifications.

3.2 Teamwork/Ada

Teamwork/Ada [4] is based on the Ada Structure Graph (ASG) editor and on the Teamwork/Ada Source Builder. The former can be used for creating models of Ada application systems using graphic icons that map to the semantic of the Ada language. The latter can be used to automatically generate source code from analyzable sets of ASGs to which appropriate source code notes shall have previously been associated.

4 INTEGRATED-PARALLEL APPROACH

The integrated-parallel approach is illustrated in part (b) of fig. 1. After a requirements analysis phase in which some preliminary formal specifications of the subcomponent to develop have been derived, the real development activities started applying both Temwork/Ada and RAISE as follows:

- Teamwork/Ada was used to represent graphically the modular decomposition of the Ada software;
- RAISE was applied to specify each identified module, as a substitution of the PDL (Program Design Language) generally used in the detailed design phase.

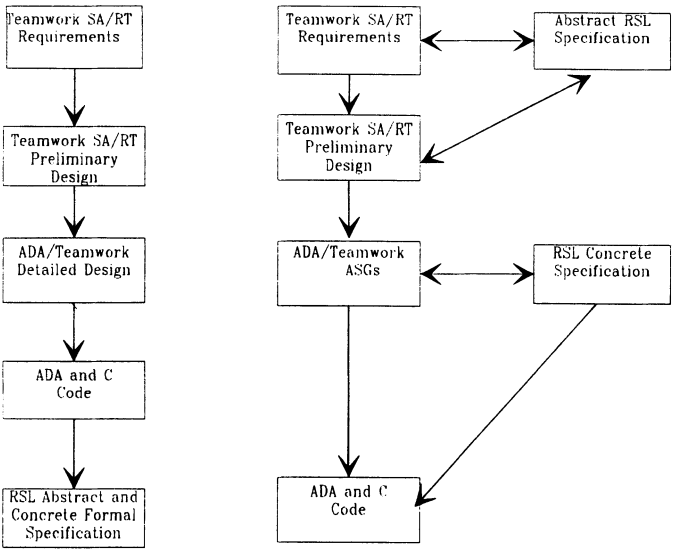

Figure 1 The after-the-fact (a) and parallel-integrated (b) approaches

5 AFTER-THE-FACT APPROACH

The after-the-fact approach is illustrated in part (a) of fig. 1. After the code development, some basic properties of the implemented alorithms were specified and verified. The specification process in itself, more than the formal proofs (which are more difficult and costly) was useful because it:

- allowed to detect some hidden errors;
- allowed to formulate the requirements in a more domain-oriented way;
- provided suggestions for reusable design as a basis for developing an Air Traffic Control software library.

5 EVALUATION OF THE QUALITY FACTORS

5.1 Early error discovery

Early error discovery is an important quality factor because, as well known, correcting an error later in the lifecycle has higher costs. One of the most severe error typology is related to misunderstanding in the requirements, because the subsequent corrections force the developer to go back to several phases of the software development.

In the TCA application, no error related to requirements misunderstaning was discovered neither in the unit nor in the SSI system test phases in the parts developed with the RAISE FM, while in the parts developed only with Teamwork/Ada a few such errors occurred. Even though direct conclusions cannot be derived from this fact, it is highly probable that the development of preliminary RSL specifications, has allowed a deeper understanding of the requirements. This conclusion is also confirmed by the fact that during the requirements analysis phase an higher number of questions arose from the specification development constituting a good basis for further clarifications with the customer.
The histogram reported in fig. 2 shows the classification of the discovered errors both for the Detector CSC, developed with formal and structured methods, and for the CSCs developed only with structured methods.

5.2 Effectiveness of communication with the customer

This quality factor, even though considered crucial to project success, because it allows to minimize misunderstanding on requirements and customer needs, could be assessed only to a minimum extent. This because:

- The design reviews normally focus on high level design choices, while in the TCA SSI application RSL was used for the detailed design of each module;
- An ad-hoc training on the specification language is required for the customer.

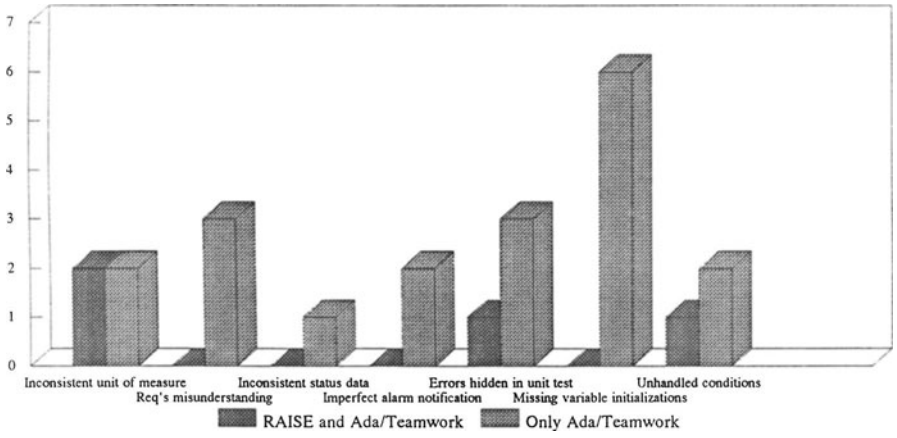

Figure 2 Error typology

5.3 Easiness in maintenance and consistency between phase products

The easiness in maintaining consistency among different lifecycle phase products is a quality factor because it provides the guarantee that the aimed consistency between detailed design and code exists. RAISE provides support to get this consistency. In the SSI application, it was generally very easy to keep consistent the detailed design and the Ada code because of the automatic Ada code generator. However the development of automatically translatable

specifications requires more effort during the detailed design phase because (a) the mapping between RSL entities and correspondent Ada code has to be checked in order to be sure that the implemented translation strategy is efficient, (b) it is necessary to keep separate the modules that shall be coded manually from those that can be automatically translated.

5.4 Compliance with process standards

The compliance with process standards is a "must" for the critical projects. For the TCA development the applicable standards were: the 2167-A DoD standard and the SSI SQS with a waterfall lifecycle model. The main need was to calibrate the 2167-A standards to fit with the adoption of RAISE for what concerns the phase products to be reviewed. As the TCA application started from the requirements analysis phase, the phase products related only to the design and coding phases. However, as the requirements phase is the one in which the applications would mostly benefit from FMs adoption, adequate guidelines should be derived as well.

5.5 Correctness of final software

During the acceptance phase no functional errors were discovered in the subcomponent developed with RAISE. In the parts developed only with structured methods, some errors were discovered. Among them, a not very severe one was related to state inconsistency between air conflict information and tracks data, allowing in particular the existence of conflicts in the related database, when the corresponding tracks had already been deleted. This typology of error might have been detected sooner with a formal and rigorous approach to verification. In fact similar errors were discovered with the after-the-fact approach.

5.6 Readibility,completeness,consistency of the final documentation

The quality of the final documentation is typically measured in terms of attributes such as clarity, consistency, accuracy, reusability. The main difference of applying FMs as SSI did in the TCA application, with respect to a traditional approach, is that the specification language substitutes the PDL with the following advantages:

- The formal specifications are internally consistent (at least from a syntattical point of view)
- If the specification language is translatable, also the consinstency between design and code is guaranteed.

On the other side, as reported in [6], the automatic code is prolific in lines of code and less readible.

5.7 Compliance with the product standards

A general observation that can be made regarding this point is that the 2167-A standard does not refer to formal specifications at all, while the ESA PSS-05-0 and PSS-01-0 mention some formal specification languages. Therefore, as usual the standards are one step behind practical usage because and some guidelines are missing.

Specifically concerning the 2167-A standard two types of problems occurred:

- Tool integration for the automatic generation of the design documentation;
- Need to define a specific outline to comment the formal specifications (see appendix A to this article).

5.8 Reusability

From the TCA application development it was evident that the application of the after-the-fact approach potentially enhances the oportunities for reusability of the produced documentation because of the more abstract and property oriented formulation of the requirements, highly independent from implementation details.

6 GUIDELINES FOR THE INTEGRATION

Even though this article does not provide detailed guidelines for applying RAISE and Teamwork/Ada with a parallel-integrated approach, it intends to mention the aspects that need to be addressed when such approaches are adopted: (a) selection of the critical component to be developed with FM (b) definition of a mapping between FM and structured method entities, (c) definition of an appropriate documentation outline to comment the formal specifications, (d) preparation of unit test plans based on the formal specifications, (e) calibration of the project applicable standards to the FM to be applied, (f) training of the customer and his/her involvement in the verification process as soon as possible.

7 CONCLUSIONS

The RAISE FM has been applied to develop a CSC part of the Analyzer-TCA software. This article has illustrated the application results, focusing on the obtained benefits. The most important lessons learnt from this experience have been:

- The formal specification process cannot avoid taking into account the training and backgrounds of those who are to read and review the specifications (testers, quality assurance responsibles, customer)
- It is a good solution to integrate the formal specifications with graphical, symbolic and tabular notations to facilitate their understanding and reviews from non-experts.
- The creation of a mathematical model of the system to develop, also when it does not represent a "deliverable", allows to obtain a "mental control" of the application that is an optimum basis to develop highly correct software.

In conclusion Formal Methods can provide some benefits to the software development process and product, but it is crucial to integrate them properly into the standard project development frameworks and acquire a more mathematical mental attitude towards software development.

8 ACKNOWLEDGEMENTS

We would like to thank the SSI Technical Manager, Piero La Vopa, for his commitment to the paper underlying activities and technical advice. Moreover we would like to thank the SSI Advanced Technology Line Manager, Matteo Piemontese, and the SSI Quality Assurance Line Manager, Domenico Dininno, for their technical review of this article.

9 LIST OF ACRONYMS

CSC	Computer Software Component
FM	Formal Method

PDL	Program Design Language
RAISE	Rigorous Approach to Industrial Software Engineering
RSL	RAISE Specification Language
TCA	Traffic Conflict Alert

10 REFERENCES

[1] The RAISE Language Group (1992). The RAISE Specification Language, edited by Prentice Hall

[2] The RAISE Method Group (1994). The RAISE development method, edited by Prentice Hall

[3] The RAISE Tool Group (1994). RAISE Tools Reference Manual.

[4] Cadre Technologies Inc. (1990) Teamwork/Ada -- User's Guide

[5] Richard A. Kemmerer (September 1990) Integrating Formal Methods into the Development Process, in *IEEE Software*,

[6] A. Alapide, M.Cinnella, P. La Vopa (1992) Automatic Generation of Ada Code with the RAISE Formal Method in *Proceedings of the third Symposium Ada in Aerospace*, Wien.

[7] A. Alapide, M.Cinnella, P. La Vopa (1992) The Viability of Applying COCOMO to RAISE-Ada Projects in *Proceedings of the third Symposium Ada in Aerospace,* held in Wien.

[8] Jeannette M. Wing, Amy Moormann Zaremski (1991) Unintrusive Ways to Integrate Formal Specifications in Practice, in *Proceedings of the 4th International Symposium of VDM Europe* Noordwijkerhout, The Nertherlands, October 1991.

[9] M.Cinnella, S.Candia, A.Alapide, S.Quaranta, D.Stellacci, P.La Vopa (1995) SSI Confidential Assessment Report on the TCA Application, *SSI report for the CEC*, doc ref. LACOS/SSI/MCSCAADS/1/V3

[10] Semmens, France and Docker (1992). Integrating Structured Analysis and Formal Specification Techniques, in *The Computer Journal*, Volume 35, Number 5.

11 BIOGRAPHY

Angela Alapide is a software engineer, graduated in Mathematics, with over six years of experience in the area of the application of formal methods to the development of high quality industrial systems. Her expertise ranges from project management to training courses organization and documents reviewing.
She represents SSI in the CEC sponsored Formal Methods Europe steering committee and regularly attends events related to the Formal Methods Technology.
Mrs. Alapide works in the SSI Advanced Technologies Department. She has coordinated the SSI dissemination of knowledge and experiences on formal methods within Alenia, the SSI mother company. (e-mail: alapide@ssi.it)

Sante Candia has over six years of experience in software development of scientific applications. After university he attended a two-year specialization course in Industrial and Applied Mathematics. He acquired experience of various software development methodologies applied throughout the software lifecycle phases. In particular, he got experience in the development of software for real time applications (satellite on-board, air

traffic control software), and for discrete and continuous simulation software. (e-mail: candia@ssi.it)

Maddalena Cinnella has been the project manager of the TCA application. She has experience in the management of projects applying formal methods in combination with more traditional methodologies. Her interests and expertise range from the specification and analysis of real-time systems to the management of critical software projects developed with advanced technologies.
She is now in charge for the activities related to the simulation of the Space Station Control Center for a Prototype of the Distributed Execution Level Planning.
She received a degree in Computer Science from the University of Bari and a Master in Business Administration. (e-mail: cinnella@ssi.it)

Sallustio Quaranta has over seven years of experience in software and algorithm development for industrial and scientific applications. He acquired experience with different software development methodologies from the traditional to the formal ones. He was involved in the Presentation Monitoring System project as lead engineer.
He received a degree in Phisics at the University of Bari. (e-mail: quaranta@ssi.it)

APPENDIX: OUTLINE OF DOCUMENTATION

10 Introduction

10.1 Scope and Purpose
10.2 Document Organization
10.3 References
10.4 Formal Development Strategy

20 Abstract Specifications

20.1 List of RSL constructs used in the abstract specifications
20.2 Abstract Specifications
    20.2.1 Module 1
    ..
    20.2.n Module n

30 Detailed Design: RSL Concrete Specifications

30.1 List of RSL constructs used in the concrete specifications
30.2 Concrete specifications
    30.2.1 Module 1
    ..
    30.2.n Module n

40 Proofs and Justifications

60 Basic Concepts on RAISE

70 Description of RSL constructs in the abstract and concrete specifications

80 Glossary

90 Analytical Index of the ASG entities

100 Cross reference ASGs/RSL specifications