Xspin/Project - Integrated Validation Management for Xspin

Theo C. Ruys

Faculty of Computer Science, University of Twente.
P.O. Box 217, 7500 AE Enschede, The Netherlands.
ruys@cs.utwente.nl

Abstract. One of the difficulties of using model checkers "in the large" is the management of all (generated) data during the validation trajectory. It is important that the results obtained from the validation are always reproducible. Without tool support, the quality of the validation process depends on the accuracy of the persons who conduct the validation. This paper discusses Xspin/Project, an extension of Xspin, which automatically controls and manages the validation trajectory when using the model checker Spin.

1 Introduction

In the past years, we have been involved in several industrial projects concerning the modelling and validation of (communication) protocols [4, 19]. In these projects we used modelling languages and validation tools - like Promela and Spin [7, 9] - to specify and verify the protocols and their properties. During each of these projects we encountered the same practical problems of keeping track of various sorts of information and data, particularly:

- many documents, which describe parts of the system, often originating from different parties;
- many versions of the same document;
- many versions of validation models, not only revisions but also variants (different abstractions of the same system);
- validation data, including:
  - simulation traces;
  - directives and options to build particular verifiers;
  - verification results;
  - counterexamples; and
  - notes and remarks on the validation runs.

The first two sources of information are mainly related to the modelling of a system whereas the latter sources of information are prominent in the validation phase of a system. We experienced that apart from the inherent state space explosion of the model of the system under validation, the validation engineer

has to deal with the data and version explosion of the modelling phase and the validation phase as well.

In [18] we suggested to use literate programming techniques [12] to tackle the management problems in the modelling phase. We also suggested that Software Configuration Management [1, 11] tools are probably needed to manage the validation phase. This paper discusses Xspin/Project, an extension of Xspin, which manages and controls the validation trajectory when using the model checker Spin.

In Sect. 2 the management problems of the validation phase are discussed. Section 3 describes Xspin/Project and the paper is concluded with Sect. 4.

In the literature on formal methods, the terms validation and verification do not have a fixed meaning. In this paper both terms are used and the following interpretations are distinguished:

- With *verification* we identify the verification process using a model checker (e.g. Spin);
- We use the term *validation* to address the controlled, systematic analysis of systems. With respect to Spin, validation includes both the simulation and verification activities.

2 Validation Management

This section discusses the management of validation data. After briefly discussing the management problems of the validation phase, the current management support within Xspin is discussed. Software Configuration Management systems are proposed as the solution to the management problems.

2.1 Validation data

In current research on automatic verification tools, much effort is being put into efficient verification algorithms whereas control and management issues are - at best - supported in a limited way. As long as nasty errors are being exposed, this may be satisfactory enough. However, when one is aiming at the systematic verification of a system, one needs more than just a smart debugging tool.

One of the practical problems when using model checkers is the management of all (generated) data during the validation trajectory. It is important that the validation results obtained using a validation tool are always reproducible [8]. Without tool support, the validation engineer has to resort to general engineering practices and record all validation activities into a logbook. Consequently, the quality of the validation process depends on the accuracy of the validation engineer. This is clearly undesirable.

When an error is found in (one of) the model(s) of the system under validation, the model(s) should of course be corrected. Furthermore, all models which have been verified previously and which are affected by the error should be re-verified. It is tedious and errorprone to re-validate all previous models and properties manually. Both the logging and the re-verification activities should be automated and - ideally - be integrated into the validation tool.

2.2 Current management support in Xspin

Since version 3.1.2, Xspin includes a "LTL Property Manager", which stores the following information on a LTL verification run into a single file with extension `.ltl`:

- the `never` claim that is generated from the LTL property;
- definitions (`#define`s) of the propositions that are used in the LTL property (and in the corresponding `never` claim);
- user provided notes; and
- the output of the verification run.

The `.ltl` file uses `#ifdef` constructs[1] to isolate the Promela fragments from the user provided notes and the verification results. Consequently, the file is liable to being updated every time a verification run is executed: previous verification results will be overwritten unless the user saves each verification run in a different `.ltl` file.

Although the "LTL Property Manager" is clearly a step in the good direction with respect to a controlled verification trajectory, some essential ingredients of the verification run are not recorded:

- the options to come to the verification results:
  - options to Spin to generate the `pan` verifier;
  - options to the C compiler to compile the `pan` verifier; and
  - options to the `pan` verifier to steer the verification run.
- the Promela model on which the verification was performed;
- the trace (trail) to the counterexample, in case the property was violated.

The last aspect becomes even more apparent when the property that is being checked requires the existence of a counterexample to be satisfied. This is the case for all existential LTL properties of the type "does there exist an execution path where P becomes true?". The reason for this is that when checking a LTL property Q, Spin will implicitly check Q for *all* execution paths. Consequently, a property like "does there exist an execution path where P becomes true" is not expressible in LTL. Instead one has to resort to check that "for all execution paths, P is always *not* true" (i.e. in LTL: $\Box\,!P$). If this transformed property is not valid, Spin will find a counterexample showing the execution path where P will become true. In this case the counterexample is the proof of the original property.

Spin is appraised most for its model checking capabilities. Besides these verification features, Spin includes a very helpful simulator, that can be used for debugging, sanity checks, rapid prototyping, simulation of generated counterexamples, etc. It is remarkable that Xspin has some limited support for verification management but has none for simulation.

[1] Promela source text is (by default) preprocessed by the standard C preprocessor, named `cpp`, before being parsed by Spin itself. The `#ifdef` directive is normally used for the conditional compilation of source text (i.e. C).

Although Xspin's "LTL Property Manager" is more than other validation tools have to offer, it is too primitive to use in an extensive validation project spanning more than a few days. Within such projects, we experienced that the validation process was seriously hampered by the fact that we had to record all our validation activities with Spin by hand. Instead of concentrating on the validation process, we had to invent schemes to keep track of the various validation models, the simulation traces, the verification results, etc. We also tried to use literate techniques to structure the validation process [18], but had to conclude that these techniques do not scale up for larger projects.

2.3 Software Configuration Management

The problems of managing the data that is generated during the validation phase relate to the maintenance problems found in software engineering [17]. This is not surprising as, in a sense, validation using a model checker involves the analysis of many successive versions of the model of a system. To tackle these maintenance problems within software engineering a lot of research has been carried out in the area of so called "Software Configuration Management".

Software Configuration Management (SCM) [1, 11] is the software engineering discipline of managing the evolution of large and complex software systems [21]. From the IEEE Standard Glossary of Software Engineering Terminology (Standard 729-1983 [10]):[2]

> Software Configuration Management is the process of identifying and defining the items in a system, controlling the release and change of these items throughout the life-cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items.

The items that comprise all information produced as part of the software engineering process are collectively called a *software configuration*. A general SCM system has the following operational aspects to manage the software engineering process [5, 10, 17]:

- *Identification.* An identification scheme reflects the structure of the product, identifies components and their type, making them unique and accessible in some form.
- *Version control.* Version control combines procedures and tools to manage different versions of configuration objects that are created during the software engineering process.
- *Change control.* Change control combines human procedures and automated tools to provide a mechanism for the control of change.

[2] SCM is a widely used term, with an equally wide range of meanings. See http://www.enteract.com/~bradapp/acme/scm-defs.html for an extensive list of alternative SCM definitions.

– *Audit and review.* Validating the completeness of a product and maintaining consistency among the components by ensuring that the product is a well-defined collection of components.
– *Reporting.* Recording and reporting the status of components and change requests, and gathering vital statistics about components in the product.

Naturally, a SCM system should be supported by automated tools. Tools for version control and build management are essential. Furthermore, SCM tools should provide the developer with a "sandbox" environment: a consistent, flexible and reproducible environment to compile, edit and debug software [13]. SCM tools have greatly evolved over the last twenty years. Tools have gone from file oriented versioning utilities to full blown repository-based systems that manage projects and support team development environments, even across geographic locations. In this paper, a discussion on particular SCM tools is clearly out of scope. The interested reader, however, is invited to visit the "Configuration Management Yellow Pages" page on Internet [22] or consult the Proceedings of the Annual Workshops on Software Configuration Management. Conradi and Westfechtel [3] give an extensive overview of the current state of art of SCM and SCM systems.

Definitions Below we define the conceptual framework for the rest of this paper, borrowing terminology from the SCM community, in particular [3, 21].

– *object.* An object (or item) is any kind of identifiable entity put under SCM control.
– *version.* A version represents a state of an evolving object.
– *revision.* A version intended to supersede its predecessor is called a revision (historical versioning).
– *variants.* Versions intended to coexist are called variants (parallel versioning).
– *configuration.* A configuration is a consistent and complete version of a composite object, i.e. a set of object versions and their relationships.
– *product space.* The product space is composed of the objects and their relationships. The product space is organized by relationships between objects, e.g. composition relationships and (build) dependency relationships.
– *version space.* The version space is composed of the set of versions. The version space is often organized into a version graph or version grid.

2.4 SCM and Xspin

It is clear that the functionality of SCM systems and tools can be of considerable value to control and manage the validation phase. For the validation trajectory using Xspin we are mainly interested in the identification of validation objects, version control over these validation objects and reporting facilities on these objects. The change control functionality and the support for audit and review supported by SCM seem less applicable to validation.

During the validation phase, a *validation object* records the results of a validation activity. For validation using Xspin, three validation objects can be distinguished: the Promela model, the property and the validation result. Of these objects, several versions exist during the validation phase.

- *Model M.* M is the model of the system under validation. M_i denotes the i-th version of model M. A version can either be a variant or a revision. Within the validation framework, variants correspond to different abstractions of the same model M, whereas revisions are different versions of the same abstraction.
- *Property ϕ.* A property ϕ is a property which should hold for the model M under verification.
- *Validation results R.* R is a set of validation results. The set R_i denotes the set of validation results obtained by executing the validation tool on the model M_i. An element from the set R_i is denoted by $r_{i,j}$. Every element $r_{i,j}$ contains the outcome (e.g. a file) of the j-th validation on M_i and additional information that depends on the type of validation. For instance, when $r_{i,j}$ corresponds to a simulation run, the simulation goal and observations on the simulation should be added to $r_{i,j}$. Whereas for a verification run, apart from the verification goal (e.g. a LTL property), the directives and options to obtain the verifier should be added to $r_{i,j}$.

The *versioned product space* of the validation phase now consists of all versions of all validation objects during the validation of the model M.

3 Xspin/Project

In this section the Xspin/Project tool is discussed. Xspin/Project is an extension of Xspin using the version control system PRCS [15]. Xspin/Project controls and manages the validation activities when using Xspin. First the choice for the underlying version control system PRCS is motivated. Then the architecture and the functionality of Xspin/Project are discussed.

3.1 PRCS

To integrate management facilities into a validation tool like Xspin, the functionality of full-blown state-of-the-art SCM tools is not needed. For a controlled and reproducible validation phase, version control and build-management are most important. A file based version control tool like RCS [20] in combination with a basic build-management tool like `make` [6] appeared to be sufficient for a prototype version of Xspin/Project.

Concurrent Version System (CVS) [2] - the de-facto version control system among free systems - seemed to be unnecessarily complex with respect to operation, administration and user interface to be easily integrated into Xspin. The author was attracted by the simplicity of PRCS and decided to use this version control system for a first prototype version of Xspin/Project.

PRCS - the Project Revision Control System [15] - is a version-control system for collections of files with a simple operational model, a clean user interface and high performance. PRCS is freely available from [14]. The current version of PRCS is implemented using RCS [20] as its back-end storage mechanism.

PRCS has some additional features which makes it well suited for integration into Xspin:

- *Conceptually close to validation objects.* PRCS defines a project version as a labeled snapshot of a group of files, and provides operations on project versions as a whole. Thus a project version naturally relates to a specific validation model and all its validation results.
- *Version naming scheme.* PRCS' version naming scheme (see below) corresponds closely to the version concepts from the validation framework: abstraction and revisions of these abstractions.
- *Simple operational model.* In PRCS, each project version is identified by a single distinguished file, the version descriptor; this file contains a description of the files included in that particular version. Adding files (i.e. validation results) only involves adding the filename to this version descriptor file.

Terminology A *project* in PRCS is a collection of (project) versions.[3] A *version* is a snapshot of a set of files arranged into a directory tree. Every version has a name of the form $m.n$, where m is the major version name and n is the minor version name. A major version name m is a string chosen by the user, whereas the minor version name n is a positive integer, assigned consecutively by the system. A PRCS *repository* contains a group of projects. Two basic operations are available to save and load versions to and from the repository respectively:

- *checkin*: a complete version is put into the repository;
- *checkout*: reconstructs a complete version, identified by the project and version name.

The PRCS concepts correspond nicely with the concepts from the validation framework. A project corresponds with the complete validation trajectory of a system. Each version of the project is a Promela model M_i together with its validation results R_i. In a version $m.n$, the major version name m corresponds with the particular abstraction of the model M and the minor version name n corresponds with the n-th revision of the particular abstraction. The fact that the major version name m in PRCS is an arbitrary string can be used to give appropriate names to the different abstraction models.

3.2 Architecture

Figure 1 shows the architecture of Xspin/Project. Xspin/Project is an extension of Xspin. The Project-part of Xspin/Project is responsible for collecting the Promela

[3] A PRCS project corresponds to the term configuration of SCM.

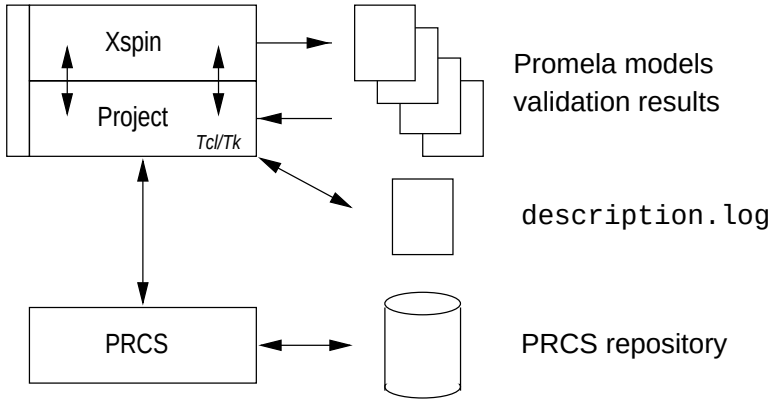

Fig. 1. Architecture of Xspin/Project.

models and validation results from Xspin and passing them to PRCS. Furthermore, the Project-part integrates a visual front end to PRCS into Xspin. The Project-extensions are written in Tcl/Tk [16].

Every Promela model M_i can be saved into the PRCS repository. Furthermore, the contents of any message box of Xspin which is the result of some validation run (i.e. $r_{i,j}$) can be saved into the PRCS repository. Xspin/Project uses a special file, i.e. `description.log` in which it stores additional information about the validation files (e.g. validation goals, options, directives, timestamps) into the current version of the project.

Xspin/Project needs PRCS version 1.2 [14] to be installed. PRCS on its turn needs RCS version 5.7 as its underlying version control system. Xspin/Project is available from http://www.cs.utwente.nl/~ruys/xspin-project.

3.3 Overview

In a nutshell the current version of Xspin/Project can be characterized as follows:

- Xspin/Project implements a visual front end to PRCS into Xspin. To the user, Xspin/Project is presented as a conceptual database of Promela models together with their validation results.
- The user of Xspin/Project can save all its validation activities into the PRCS database. Furthermore, the user is given the possibility to annotate these validation activities.
- All essential verification data such as directives and options to the C compiler and the `pan` verifier are automatically saved into the PRCS repository.
- Xspin/Project ensures the integrity of the Promela models and their validation models.

Xspin/Project uses plain PRCS as its underlying configuration management tool. This means that all additional powerful features (like `diff` and `merge`) of PRCS

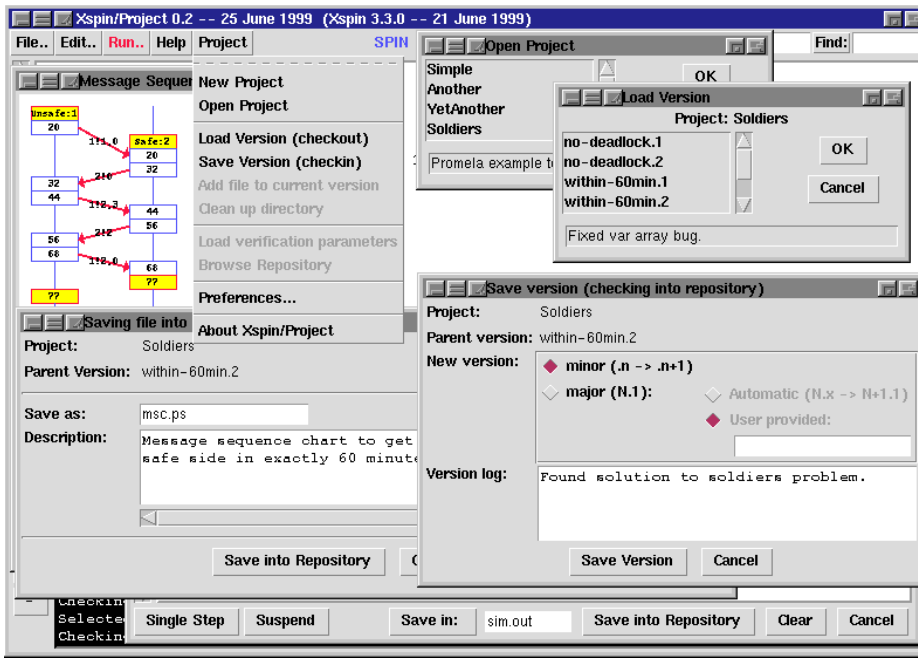

Fig. 2. Screen capture of a validation session with Xspin/Project.

are also available to the user. However, these advanced features of PRCS are not (yet) available from within Xspin/Project. To exploit these features, one should use PRCS' command-line options.

3.4 User awareness

Figure 2 captures a screenshot of a validation session with Xspin/Project. The added functionality of Xspin/Project provides the user with a "sandbox" environment: a consistent, flexible and reproducible environment to edit and validate Promela models. The user should not be unnecessarily hampered during the validation trajectory. Below we discuss the user awareness with respect to the features added by Xspin/Project on top of the original Xspin.

– *Accessing PRCS.* An extra toplevel menu has been added to Xspin: Project. This menu can be used to access most Xspin/Project functions, like:
  - Starting a new project.
  - Opening an existing project.
  - Loading (checking out) a particular Promela model (i.e. an explicit version of the project).
  - Saving (checking in) a particular Promela model and all its recorded validation results.

- Adding files explicitly to the current version. This may be useful when non-Xspin files are relevant to a validation run or when one has forgotten to save a Xspin file into the repository.
- Cleaning up the directory. Using this function all files that have been saved previously in the repository are removed from the current directory.

– *Saving validation results.* To every dialog box containing validation output (e.g. simulation traces, message sequence charts) an extra button has been added: "Save into Repository". When this button is pressed, Xspin/Project shows a dialog box where the user can annotate the particular file with some notes on the particular validation run. The file and the (optional) notes are subsequently saved into the repository. Furthermore, for verification runs, Xspin/Project saves all options that are needed to build and run the `pan` verifier into the `description.log` file.
– *Forcing version integrity.* When the user has saved the results of a validation run into the current version of the project, the corresponding Promela model will be locked: the user can only perform additional validation runs on the model. Only when the complete version has been saved (checked in) into the repository, the Promela model will be unlocked again for user edits. This strategy of Xspin/Project is necessary to keep all models and their validation results accurate.[4]

Software development vs. validation When using a SCM tool to control the software development process, a version of the 'product' consists of several files and rules to construct the product. Older versions correspond to inferior or less stable versions (containing bugs) or to versions of the product with fewer features.

In Xspin/Project, PRCS is used as a database to store and log all validation activities. Each different Promela model is stored together with the validation results on that particular model. In contrast with software development, earlier versions of the model are not inferior or less stable versions, but should be considered as different abstractions of the same model.

4 Conclusions

The success of model checking tools is mainly based on the bugs and errors that those verification tools have exposed in (existing) systems and standards. Now that model checking tools are becoming more widespread, the application of model checkers is slowly shifting from debugging to verification.

[4] This strict behaviour does not restrain the user when constructing a new Promela model. During the development of a Promela model, one usually performs several sanity checks (mostly simulation runs) on intermediate models before actual verification runs are tried. Naturally, these sanity runs do not have to end up in the validation repository. Therefore, the user is not forced to save all validation files but may only optionally do so.

This paper discusses the need for systematic control and management over the (generated) data when using an analysis tool like Spin for the validation of large systems. The strength of SCM systems and tools has briefly been discussed. We have concluded that the full power of SCM systems is not needed to manage the validation activities; a flexible version control mechanism is sufficient to manage the validation phase when using Xspin.

We have presented Xspin/Project, an integration of the version control system PRCS into Xspin. The current version of Xspin/Project presents the user with a conceptual database for Promela models and their validation results. To guide the verification engineer even further, we are currently working on the following extensions to Xspin/Project:

- *Reporting*: adding reporting facilities to generate a detailed overview of the complete validation trajectory.
- *Reverification*: when a Promela model has been altered, all previous verification runs on the model should be automatically re-verified.
- *Reuse*: reusing verification options of previous verification runs to verify new versions of Promela models.
- *Compare*: comparing different versions of Promela models (using PRCS' `diff` command) to get information on the abstractions and revisions made during the validation trajectory.

But even without these additions the current version of Xspin/Project already promises to be a great help in managing the version space explosion.

References

[1] Wayne A. Babich. *Software Configuration Management: Coordination for team productivity.* Addison-Wesley, Reading, MA, 1986.

[2] Brian Berliner. CVS II: Parallelizing Software Development. In *Proceedings of the Winter 1990 USENIX Conference, January 22-26, 1990, Washington DC, USA*, pages 341–352, Berkeley, CA, USA, January 1990. USENIX.

[3] Reidar Conradi and Bernhard Westfechtel. Version Models for Software Configuration Management. *ACM Computing Surveys*, 30(2):232–282, June 1998.

[4] Pedro R. D'Argenio, Joost-Pieter Katoen, Theo C. Ruys, and G. Jan Tretmans. The Bounded Retransmission Protocol must be on time! In Ed Brinksma, editor, *Proceedings of the Third International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97)*, number 1217 in Lecture Notes in Computer Science (LNCS), pages 416–431, University of Twente, Enschede, The Netherlands, April 1997. Springer Verlag, Berlin.

[5] Susan Dart. Concepts in Configuration Management Systems. In P.H. Feiler, editor, *Proceedings of the Third International Workshop on Software Configuration Management (SCM'91)*, pages 1–18, Trondheim, Norway, June 1997. ACM SIGSOFT, ACM Press, New York.

[6] Stuart I. Feldman. Make – A Program for Maintaining Computer Programs. *Software – Practice and Experience*, 9(3):255–265, March 1979.

[7] Gerard J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[8] Gerard J. Holzmann. The Theory and Practice of a Formal Method: NewCore. In *Proceedings of the IFIP World Congress*, Hamburg, Germany, August 1994. Also available from URL: http://cm.bell-labs.com/cm/cs/doc/94/index.html.

[9] Gerard J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997. See also URL: http://netlib.bell-labs.com/netlib/spin/whatispin.html.

[10] IEEE. *IEEE Standard Glossary of Software Engineering Terminology: IEEE Standard 729-1983*. IEEE, New York, 1983.

[11] IEEE. *IEEE Guide to Software Configuration Management: ANSI/IEEE Std 1042-1987*. IEEE, New York, 1987.

[12] Donald E. Knuth. Literate Programming. *The Computer Journal*, 27(2):97–111, May 1984.

[13] David B. Leblang and Paul H. Levine. Software Configuration Management: Why is it needed and what should it do? In Jacky Estublier, editor, *ICSE SCM-4 and SCM-5 Workshops – Selected Papers*, number 1005 in Lecture Notes in Computer Science (LNCS), pages 53–60. Springer Verlag, Berlin, 1995.

[14] Josh MacDonald. PRCS – Project Revision Control System. Available from URL: http://www.xcf.berkeley.edu/~jmacd/prcs.html.

[15] Josh MacDonald, Paul N. Hilfinger, and Luigi Semenzato. PRCS: The Project Revision Control System. In B. Magnusson, editor, *Proceedings of the ECOOP'98 SCM-8 Symposium on Software Configuration Management (SCM'98)*, number 1439 in Lecture Notes in Computer Science (LNCS), pages 33–45, Brussels, Belgium, July 1998. Springer Verlag, Berlin.

[16] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

[17] Roger S. Pressman. *Software Engineering – A Practioner's Approach*. McGraw-Hill, New York, third edition, 1992.

[18] Theo C. Ruys and Ed Brinksma. Experience with Literate Programming in the Modelling and Validation of Systems. In Bernhard Steffen, editor, *Proceedings of the Fourth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, number 1384 in Lecture Notes in Computer Science (LNCS), pages 393–408, Lisbon, Portugal, April 1998. Springer Verlag, Berlin.

[19] Theo C. Ruys and Rom Langerak. Validation of Bosch' Mobile Communication Network Architecture with SPIN. In *Proceedings of SPIN97, the Third International Workshop on SPIN*, University of Twente, Enschede, The Netherlands, April 1997. Also available from URL: http://netlib.bell-labs.com/netlib/spin/ws97/ruys.ps.Z.

[20] Walter F. Tichy. RCS – A System for Version Control. *Software – Practice and Experience*, 15(7):637–654, July 1985.

[21] Walter F. Tichy. Tools for Software Configuration Management. In J.F.H. Winkler, editor, *Proceedings of the International Workshop on Software Version and Configuration Control*, pages 1–20, Grassau, Germany, January 1988. Teubner Verlag.

[22] André van der Hoek. Configuration Management Yellow Pages. Available from: http://www.cs.colorado.edu/users/andre/configuration_management.html, 1999.