Annales Univ. Sci. Budapest., Sect. Comp. 37 (2012) 81–92

SOURCE CODE SCANNERS IN SOFTWARE QUALITY MANAGEMENT AND CONNECTIONS TO INTERNATIONAL STANDARDS

Anna Bánsághi, Béla Gábor Ézsiás,
Attila Kovács and Antal Tátrai
(Budapest, Hungary)

Communicated by Zoltán Horváth

(Received January 15, 2012; revised February 7, 2012;
accepted February 14, 2012)

Abstract. This paper deals with software quality by analyzing how source code scanners in quality management are connected to international product quality standards. The article has three parts: the authors (1) provide a short overview of software quality management, (2) examine to what extent can quality metrics of the static source code scanners PMD and FxCop be classified into the attribute sets of ISO/IEC 9126 and ISO/IEC 25010 quality models, (3) investigate the quality road from the code to the product via the ISO/IEC 25010 quality model.

1. Software quality management

Software quality is one of the most important aspects of a software company. Computers are being used in an increasingly wide variety of application areas, their intended and correct operation is often critical for business success and/or human safety. IT companies perform various activities and follow various strategies to ensure the quality of their software systems.

Key words and phrases: Product Quality, ISO/IEC 9126, ISO/IEC 25010, PMD, FxCop.
1998 CR Categories and Descriptors: D.2.8, D.2.9.
The Research is supported by the European Union and co-financed by the European Social Fund (grant agreement no. TÁMOP 4.2.1./B-09/1/KMR-2010-0003).

Several quality standards, models, and methods have been developed in the past to guide software companies in defining and institutionalizing their quality management processes. Approaches, like ISO/IEC 9001, CMMI[1], SPICE[2], Automotive SPICE focus on the quality of the software *process*. Some other approaches, like ISO/IEC 9126 and ISO/IEC 25010 are about software *product* quality. Others, e.g. GQM[3], describe *measurement* techniques used in software development, while even others, like PSP and TSP[4] emphasize the importance of human resources and personal processes used by the software development. Nowadays, ITIL[5], COBIT[6], SoX[7] and SQUARE[8] are becoming more and more popular, and other approaches, like Six Sigma, sometimes are also present in the same quality management environment. In this paper we focus on software product quality and we investigate the way source code scanners can be used in quality management. First, for the sake of better understanding, we survey the main aspects of software quality, the general (product) quality standards and the source code scanners PMD and FxCop.

1.1. Aspects of software quality

The notion of software quality (SQ) can be defined in multiple ways, see [1, 2, 11, 13, 22, 23]. Although the various definitions are clear and unambiguous, the concept of quality is noticeably complex. Just a few examples: Kitchenham [19] states that quality is hard to define, impossible to measure, but easy to recognize. Gilles [15] states that quality is generally transparent when present, but easily recognized in its absence. DeMarco [14] states that a product's quality is a function of how much it changes the world for the better. An important and useful approach of software quality is to consider it in a technical and in a business context [7]. Functional requirements describe the behaviors (functions, services) of the system that support user goals, tasks or activities. Based on these requirements functional quality is typically measured through specification-based software testing. The non-functional quality defines how well non-functional requirements are met that support the delivery of functional requirements. It is usually related to code and its internal structure. In this paper we consider these aspects.

[1]Capability Maturity Model Integration
[2]ISO/IEC 15504 Information Technology, Process Assessment, also known as SPICE (Software Process Improvement and Capability Determination)
[3]Goal Question Metric
[4]Personal Software Process, Team Software Process
[5]Information Technology Infrastructure Library
[6]A framework created by ISACA for IT Management and Governance
[7]Sarbanes-Oxley Act mandate controls to manage risk in the organization
[8]Requirements Engineering for Improved System Security and Privacy

1.2. Product quality standards

Specification and evaluation of software product quality is a key factor in ensuring adequate business quality. This can be attained by defining appropriate quality characteristics taking into account of the intended use of the software product [3, 5]. From a technical perspective, quality attributes drive significant architectural and design decisions. Software quality management (SQMG) helps to ensure that the required level of overall quality is achieved. The ISO 9126 standard categorizes quality from the user perspective as functionality, reliability, usability, efficiency, maintainability and portability providing 27 subcharacteristics of external quality (see Figure 1).

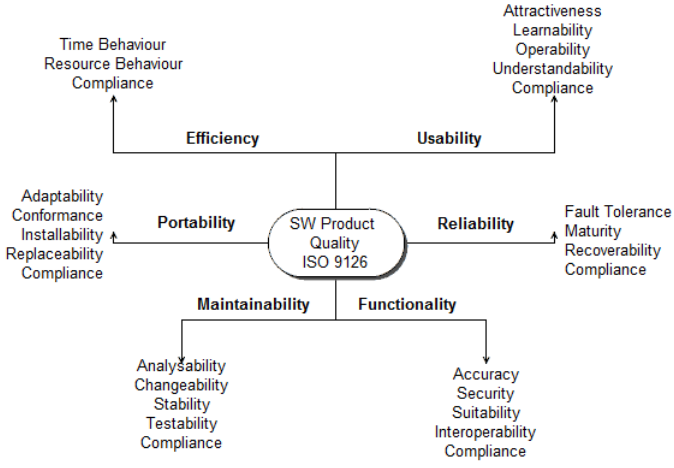

Figure 1. The software product quality based on the model ISO/IEC 9126

Quality-in-use is a combined effect of the six categories. Recently, the standard has been replaced by the ISO 25010 model (see Figure 2). This model has eight product quality characteristics and 39 subcharacteristics. The standard describes the internal and the external measures of software quality: internal measures describe a set of static internal attributes that can be measured, while the external measures focuses more on software as a black box and describes external attributes. Besides the software product quality model, the standard also describes another model, the model of software quality in use.

By comparing the two models it can be stated that the new model has a broader range and is more accurate, but it suffers exaclty the same illnesses as ISO 9126, namely, different parties with different views of software quality can select different definitions. The ISO 9126 and the subsequent ISO 25010

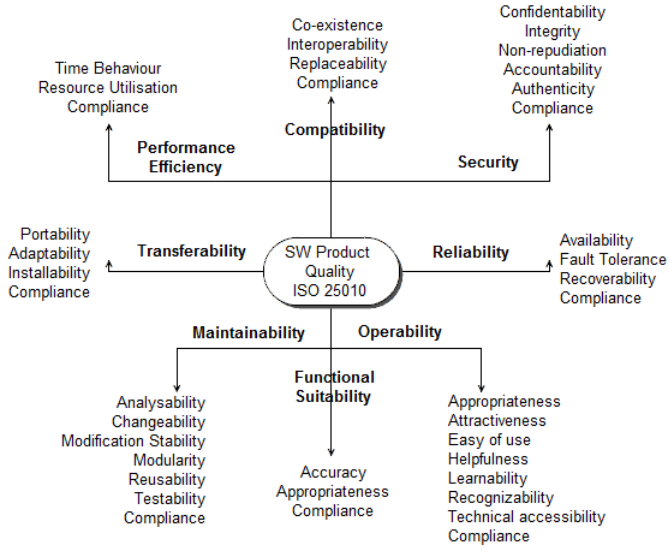

Figure 2. The software product quality based on ISO/IEC 25010 model

standards both provide a good frame of reference for software product quality, but do not offer a practically applicable method for quality assessment. On the other hand, vast literature on software metrics proposes numerous ways of measuring software without providing a traceable and in practice instantly applicable translation to the multi-faceted notion of quality. We note hovewer, that there are some valuable results in this direction, e.g. applying ISO 9126 based models for test specifications [25] or measuring maintainability via the maintainability index [16].

For the sake of completeness we note that SQuaRE (Software product Quality Requirements and Evaluation) describes the structure, classification and terminology of attributes and metrics applicable to SQMG [4]. Based on the mentioned models the internal quality characteristics have been defined by the Consortium for IT Software Quality (CISQ) [6]. CISQ has defined 5 major quality characteristics needed for a piece of software to provide business value: (1) reliability, (2) efficiency, (3) security, (4) maintainability, and (5) size.

1.3. The source code scanners PMD and FxCop

In software projects, the quality attributes have to be measured and evaluated. Software quality measurement is about quantifying to what extent a

software or system rates along each of the quality dimensions. An example for such a measurement is static analysis. It is useful for catching common errors early. Source code of various programming languages can be scanned and analyzed in order to understand the code's logical structure and determine the impact of change. Source code scanner tools (SCST) provide metrics and often statistical values for the scanned source code which can be used as the basis for specific analysis and quality evaluations. In other words, these tools are able to recognize "code smells" (after Kent Black). Code smell is a hint that something might be wrong in the code. It is important that calling something a code smell is not an attack; it is simply a sign that in order to preserve or reconstruct quality code a closer look is warranted. There are numerous SCST tools accessible [10]. In this paper we use two of them.

PMD [8] scans Java source code and looks for potential problems like possible buggy statements, dead code, unused local variables, parameters and private methods, suboptimal code, overcomplicated expressions, duplicate code, etc. PMD is integrated with numerous IDEs. At present the latest version is 4.2.5. We considered the following metric sets: Basic (33 rules), Braces (4 rules), Clone Implementation (3 rules), Code Size (11 rules), Coupling (3 rules), Design (48 rules), Finalizer (6 rules), Import Statement (5 rules), JUnit (11 rules), Logging (7), Migration (14 rules), Naming (19 rules), Optimization (10 rules), Exception (12 rules), String and StringBuffer (15 rules) and Security (2 rules). We note that there are some rules in more than one metric sets.

FxCop [9] is an application that analyzes managed code assemblies (code that targets the .NET framework common language runtime) and reports information about the assemblies, such as possible design, localization, performance, and security improvements. FxCop is intended for class library developers, designed to be fully integrated into the software development cycle, and suited for use as part of the automated build processes. It can be integrated with Microsoft Visual Studio .NET. The current version is 10.0. We considered the following rule sets: Design (62 rules), Globalization (11 rules), Interoperability (16 rules), Mobility (8 rules), Naming (24 rules), Performance (16 rules), Portability (3 rules), Security (42 rules) and Usage (43 rules). Here every rule belongs to exactly one category.

2. Classifying the quality attributes by the standards

In this section we examine to what extent the quality metrics provided by these tools can be classified into the attribute sets of the standards ISO 9126 and ISO 25010.

The classification process type was a technical review with at least 3 participants, all of them having at least a two-year project experience in the given programming language (Java experience in the case of PMD and C# experience in the case of FxCop). In the review meeting each rule has been discussed and the probability of belonging to some quality attibute set has been quantified. Each rule was allowed to belong to more than one set. If for a given attribute the average quantity was more than or equal to 90%, then the rule was accepted to belong to the appropriate set, otherwise was not. In this way, the classification is reproducible having the same results with high probability.

As an example, let us see the PMD rule TooManyStaticImports. The rationale of the rule is that if one overuses the static import feature, it can make the program unreadable and unmaintainable polluting its namespace with all the static members imported. Readers of the code (including the author) a few month later will not able to recognize (or only with extra effort) which class a static member comes from. Therefore, the rule has been classified into the Maintainability/Analyzability quality category with 100% consensus. In this way we categorized all the 203 PMD rules and 225 FxCop rules into ISO 9126 and ISO 25010 categories (Figures 3–4).

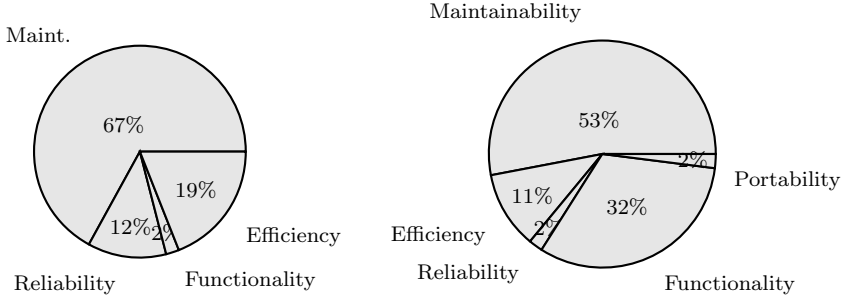

Figure 3. The distribution of the rule sets in the case of PMD (left) and FxCop (right) regarding ISO 9126 quality categories.

We note that in the case of FxCop the pre-defined categorization (naming) of the rules and rule sets gave a significant support for the classification.

Some interesting observations:

- In the case of PMD the impact of the changes in the quality categories were marginal. In contrast, the impact of the changes in the case of FxCop were more significant.
- Regarding ISO 25010 FxCop has more rules for Technical Security, Functional Suitability and Compatibility, but proportionally less for Reliability.

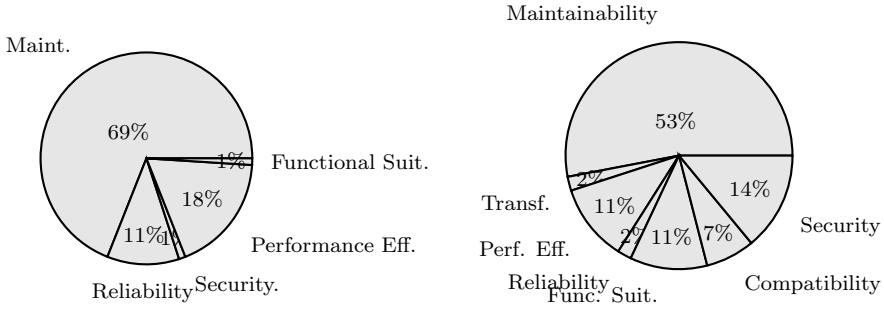

Figure 4. The distribution of the rule sets in the case of PMD (left) and FxCop (right) regarding ISO 25010 quality categories.

- Both tools have approximately the same number of rules for Performance Efficiency.
- The most rules of the applied source code scanner tools are assigned to the Maintainability attribute set.
- Inside the Maintainability characteristic the most significant subcharacteristic was Analysability. In the standard ISO 9126
  - Case PMD: Analysability has 66%, Stability has 21%, Chengeability has 10% and Testability has 3% of the Maintainability rules,
  - Case FxCop: Analysability has 76%, Stability has 6%, Chengeability has 10% and Testability has 8% of the Maintainability rules.

  In the standard ISO 25010
  - Case PMD: Analysability has 63%, Modification Stability has 21%, Chengeability has 6% Testability has 3%, Modularity has 4% and Reusability has 3% of the Maintainability rules,
  - Case FxCop: there were no significat changes to ISO 9126.

3. The quality road from code to product

In order to specify non-functional quality, various distinctions can be made. In the case of the executing system the qualities of interest are relative to user goals. We will refer to these as run-time qualities. In the case of the work products the qualities are driven by the development organization's goals. We

will refer to these as development-time qualities (see [20]). The scope of these requirements may be system-wide or local to some specific behavior.

- *Run-time quality* describes how well the functional requirements are satisfied. The attributes are useability (such as ease-of-use, learnability, recognizability, efficiency, etc.), configurability, supportability, correctness, reliability, availability, QoS requirements (such as performance, throughput, response time, transit delay, latency, etc.), safety properties (such as security and fault tolerance), operational scalability (including support for additional users or sites, or higher transaction volumes).

- *Development-time quality.* In addition to systems that satisfy their users, the development organization has interest in the properties of the artifacts of the development process (design, code, test, etc.). Qualities of these artifacts influence the effort and cost associated with the current development as well as support for future changes or uses (maintenance, enhancement, reuse). Examples of development-time quality requirements are localizability, modifiability, extensibility, evolvability (support for new capabilities or ability to exploit new technologies), composability (ability to compose systems from plug-and-play components), reusability.

Extending this idea, a three-level quality model was introduced by Plösch et al. (design quality, code quality, runtime quality) [21].

Figure 5 shows the connections and influences of code quality to product quality. Code quality is closely related to architecture quality: a quality code has to be well structured (but, of course, it is not enough). Architecture quality consists of system architecture quality, package (module) quality, class quality and abstract data structure quality. The development-time quality influences the code quality via the architecture, and the code with its architecture influences the runtime quality. Source code scanners are able to examine the architecture and the development-time quality decisions, and (to some extent) the runtime properties as well. Source code scanners provide data for measuring quality via quality rules. In our interpretation, code quality can be measured via source code scanners applying the following quality rule sets:

- Complexity,
- Exception handling,
- Comment quality,
- Logging,
- Naming conventions,

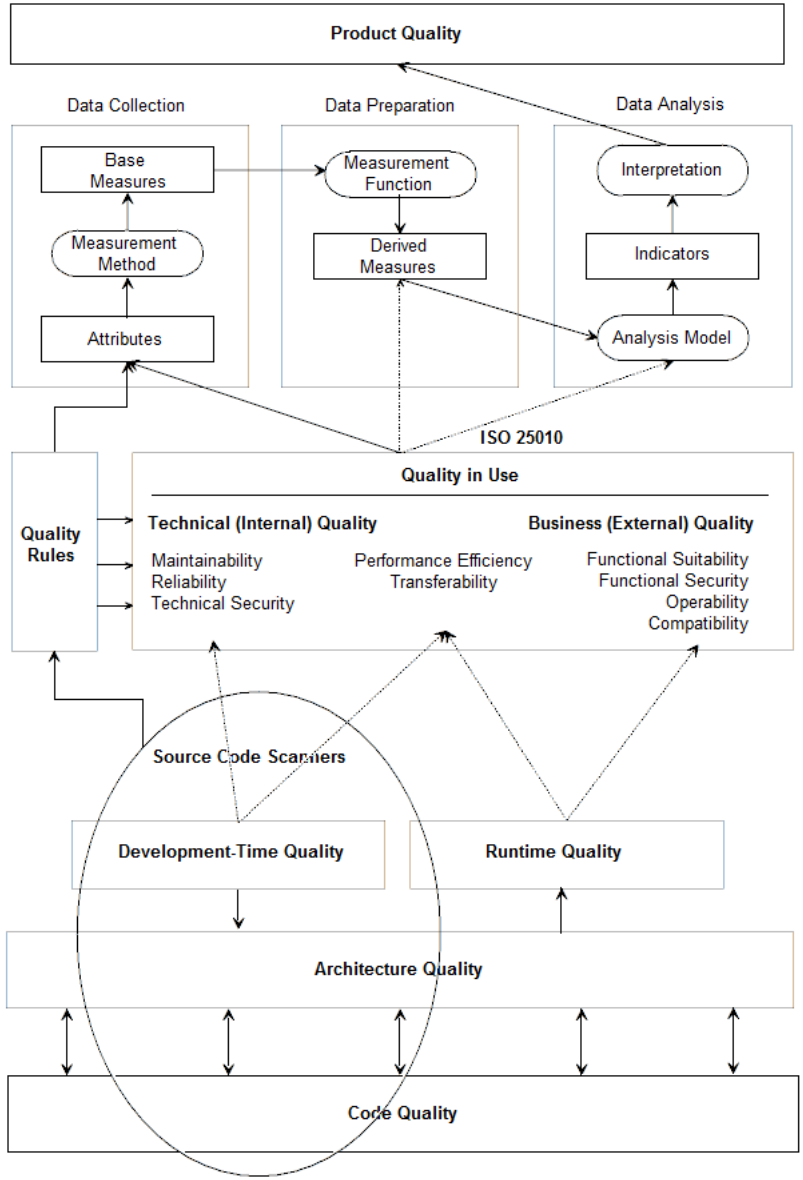

Figure 5. Source code scanners and product quality

- Security,
- Proper language style,
- Proper programming technique,
- Volume,
- Resource efficiency,
- Time efficiency,
- Code clones.

The authors should like to raise the reader's attention for the importance of testing quality as being a significant part of the development-time quality. It must be also noted that there are positive and negative influences among the quality attributes (e.g. a more efficient code may be harder to maintain, while appropriate comment quality may make it easier).

4. Conclusions and further work

In this paper we examined the ISO/IEC 9126 and ISO/IEC 25010 quality models from the viewpoint of static code analyser capabilities. We assigned the quality rules of the tools PMD and FxCop to the quality characteristics and subcharacteristics of the standards (more than 200 rules in each tool). We conclude that

- Case PMD: the classification of the rules yields almost the same distribution in the quality characteristics of ISO 9126 and ISO 25010.
- Case FxCop: the characteristic Functionality of ISO 9126 represents proportionally 32% but only 11% in the case of ISO 20510. The new category Security represents 14%, which means that FxCop stresses significant emphasis on security. The characteristic Maintainability represents the same proportion in both quality models (more than 50%).

For both standards the dominant characteristic was Maintainability with dominant subcharacteristic Analysability.

One of our future plans is to give a model for supporting the establishment and measurement of attributes in quality profiles via source code scanners and

call-graph analysis. The quality profile presents the relevant quality characteristics and the evaluation levels for the software product. It reflects the notion of quality for a certain software product and makes quality clear and measurable for both developers and users. The profile is based upon information about customer/user, business process and the software product itself. Our second plan is to analyse the impact of automatic source scanner issues on code quality.

References

[1] **IEEE 610.12:1990**, *IEEE Standard Glossary of Software Engineering Terminology.*

[2] **ISO/IEC 9000:2000**, *ISO Standard for Quality Management Systems – Fundamentals and Vocabulary.*

[3] **ISO/IEC 9126:2001-2004**, *ISO Standard for Software Engineering – Product quality – Part 1: Quality model, Part 2: External metrics, Part 3: Internal metrics, Part 4: Quality in use metrics.*

[4] **ISO/IEC 25000:2005**, *ISO Software Engineering – Software Product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE.*

[5] **ISO/IEC 25010:2011**, *ISO Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.*

[6] **CISQ**, *Consortium for IT Software Quality*, `www.it-cisq.org`, retrieved: 2012-01-01.

[7] **Wikipedia**, *Software Quality*, retrieved 2012-01-01.

[8] **PMD**, `pmd.sourceforge.net`, retrieved: 2012-01-01.

[9] **FxCop**, `msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx`, retrieved 2012-01-01.

[10] **Wikipedia, Static code analysers**, `List_of_tools_for_static_code_analysis`, retrieved 2012-01-01.

[11] *American Society for Quality, Glossary - Entry: Quality*, `www.asq.org/glossary/q.html`, retrieved 2012-01-01.

[12] **Abran, Alain, Al-Qutaish, Rafa E. and Desharnais, Jean-Marc,** Harmonization issues in the updating of the ISO standards on software product quality, *Metrics News Journal*, **10/2**, ed. Otto-von-Guericke, University of Magdeburg, Germany, December, (2005), pp. 35–44.

[13] **Crosby, Philip,** *Quality is Free*, McGraw-Hill, 1979.

[14] **DeMarco, Tom,** *Management Can Make Quality (Im)possible*, Cutter IT Summit, Boston, April 1999.

[15] **Gillies, Alan C.,** *Software Quality, Theory and Management*, International Thomson Computer Press, 1996.

[16] **Heitlager, I., T. Kuipers and J. Visser,** A practical model for measuring maintainability, in: *Proceedings of the 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, IEEE Computer Society Press, 2007, pp. 30–39.

[17] **Ho-Won Jung, Seung-Gweon Kim, Chang-Sin Chung,** Measuring software product quality: A survey of ISO/IEC 9126, *IEEE Software*, **21/5**, September/October, (2004), 10–13.

[18] **Kan, S.H.,** *Metrics and Models in Software Quality Engineering*, Addison-Wesley, Boston, MA, second edition, 2002.

[19] **Kitchenham, Barbara and Shari Lawrence Pfleeger,** Software quality: The elusive target, *IEEE Software*, **13/1**, (January 1996), 12–21.

[20] **Malan, R. and D. Bredemeyer,** *Defining non-functional requirements*, `www.bredemeyer.com/pdf_files/NonFunctReq.PDF`, retrieved 2012-1-1.

[21] **Plösch, R., H. Gruber, C. Krner, G. Pomberger and S. Schiffer,** A proposal for a quality model based on a technical topic classification, *Proceedings of SQMB 2009 Workshop*, held in conjunction with SE 2009 conference, March 3rd 2009, Kaiserslautern, Germany, published as Technical Report TUM-I0917 of the Technical University Munich, July 2009.

[22] **Pressman, Roger,** *Software Engineering: A Practitioner's Approach*, 5th Edition, McGraw Hill, 2001.

[23] **Schulmeyer, G. Gordon and James I. McManus,** *Handbook of Software Quality Assurance*, 3rd edition, Prentice Hall PRT, 1998.

[24] **Spinellis, D.,** *Code Quality: The Open Source Perspective*, Addison Wesley, Boston, MA, 2006.

[25] **Zeiss, B., D. Vega, I. Schieferdecker, H. Neukirchen and J. Grabowski,** Applying the ISO 9126 quality model to test specifications - exemplified for TTCN-3 test specifications, *Software Engineering*, (2007), 231–244.

A. Bánsághi and A. Kovács
Department of Computer Algebra
Faculty of Informatics
Eötvös Loránd University
Pázmány P. sétány 1/C.
H-1117 Budapest
Hungary
`bansaghi@inf.elte.hu`
`attila.kovacs@compalg.inf.elte.hu`

B.G. Ézsiás and A. Tátrai
MeasureIT Ltd.
Budapest
Hungary
`ezsias.bela@gmail.com`
`tatraian@caesar.elte.hu`