Determining Potential Errors in Tool Chains

Strategies to Reach Tool Confidence According to ISO 26262

Martin Wildmoser, Jan Philipps and Oscar Slotosch

Validas AG, Munich, Germany

{wildmoser,philipps,slotosch}@validas.de

Abstract. Due to failures of software tools faults compromising the safety of the developed items may either be injected or not detected. Thus the safety norm for road vehicles, ISO 26262, requires to evaluate all software tools by identifying potential tool failures and measures to detect or avoid them. The result is a tool confidence level for each tool, which determines if and how a tool needs to be qualified. This paper focuses on tool failure identification and proposes two strategies for this task. The function-based strategy derives potential tool failures from a functional decomposition of the tool. The artifact-based strategy analyzes artifacts only. We introduce an analysis tool to support these strategies and discuss their ability to produce lists of failures that are comprehensive, uniform and adequately abstract. This discussion is based on our experience with these strategies in a large scale industrial project.

Keywords: ISO 26262, Tool Chain Analysis, Tool Qualification, HAZOP, potential tool failure, potential tool error

1 Introduction

The use of software to control technical systems – machinery, aircraft, cars - carries risks in that software defects may endanger life and property. Safety standards, such as the recent ISO 26262 [1] for the automotive domain aim to mitigate these risks through a combination of demands on organization, structure and development methods. These standards and the practices they encode can also be seen as a sign of maturity, a shift from an anything-goes attitude of programming to a more disciplined engineering approach to software development.

As in any discipline, with growing maturity more emphasis is put not only on the way of working, but also on the tools used. In safety standards, we can observe a similar development. Earlier standards put only little emphasis on tool use, perhaps roughly demanding an argument that each tool be "fit for use", mainly for tools used in generating or transforming code or in testing software or systems. Recent standards, such as the ISO 26262 take a more holistic viewpoint. Not only tools, but also their use in the development process must be analyzed, risks identified and countermeasures employed. In this line of thinking, also requirement management tools,

version control systems, and the plethora of little helper tools to integrate work flows must be considered. Establishing confidence in tools according to ISO 26262 is a two-step process.

In the first step, tools are evaluated to determine a tool confidence level (TCL), based on the possibility of tool failures leading to a violation of a safety requirement, and on the likelihood of detecting or preventing such tool failures (see Fig. 1).

In the second step, depending on the TCL and the safety integrity level of the development object, further qualification measures are demanded, such as for instance extensive tests (tool validation).

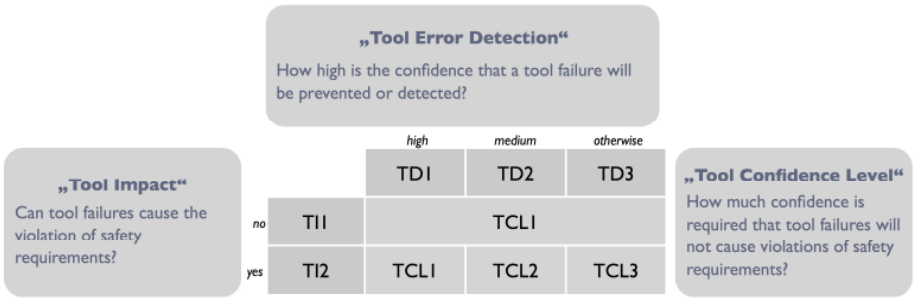


Fig. 1. Tool evaluation according to ISO 26262:2011-8

While there is a certain body of work on tool validation issues [2, 3] and some work on specific tool use patterns to reduce the need of tool validation [4], there is so far little work that considers the holistic approach of establishing tool confidence according to ISO 26262. A notable exception is [5], where the process of tool evaluation is explained in detail. The contribution of this paper is a strengthening of [5]: We follow a similar approach, but in this paper we concentrate on *determining potential tool failures* as a basis for obtaining correct tool confidence levels.

In practice the determination of the TCLs of a real development tool chain has to deal with dozens of tools and hundreds of use cases, potential tool failures and counter-measures for these. Rigorous bookkeeping is needed to obtain comprehensible and consistent results. In addition, if not done in a systematic way, the determination of potential tool failures will largely depend on the experience, biased view and chosen approach of the person carrying out the analysis.

This paper proposes strategies for systematic determination of potential tool errors and judges their comprehensiveness, uniformity, adequateness of results and scalability. This judgment is not merely theoretical but backed up by the practical experience gained by the authors from applying the proposed strategies in a large scale project where the entire development tool chain for a product family of an automotive supplier has been evaluated. The paper also introduces a tool called Tool Chain Analyzer that has been built to support tool chain evaluation.

This paper is structured as follows. In the next section, based on a simple example, we give an overview on the tool evaluation process. Section 3 contains the main contribution: we propose and discuss two strategies for identifying potential tool failures. Section 4 briefly introduces a tool to support these strategies. Section 5 concludes.

2 Tool Evaluation Process

Tool evaluation is about determining the TCL for all tools used in the development process of a safety related product. The ISO 26262 defines what a tool evaluation report must contain, but leaves the process for tool evaluation largely open. The process we currently follow for tool evaluation consists of the following steps:

1. Define list of tools
2. Identify use cases
3. Determine tool impact
4. Identify potential tool failures
5. Identify and assign measures for tool failure detection and -prevention
6. Compute tool confidence level for each use case and tool

First, we create a list of all tools used in the development process. Then by studying the development process and by interviewing tool users we identify and write down the use cases for each tool (why? who? when? what? how?). For each use case we then determine the tool impact (TI1, TI2) by answering two questions:

1. Can a tool failure inject a safety-related fault into the product?
2. Can a tool failure lead to the non-detection of a safety-related fault in the product?

Only if both questions can be answered with "No" the tool has no impact (TI1). For every use case with impact (TI2) the potential tool failures need to be identified. For each potential tool failure we look for existing measures for detection or –prevention in the development process. If such measures are found we assign them to the corresponding potential tool failure together with an estimated tool error detection level (TD1-TD3). From the TI and TD the we finally determine the TCL according to tables in ISO 26262 (see Fig. 1).

To give a short example (see Fig. 2) assume a tool chain consisting of the tools Zip, Diff and Ls, which are used for release packaging.

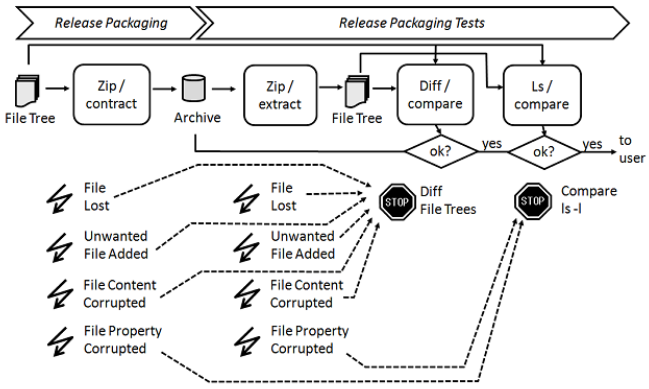


Fig. 2. Release Packaging Tool Chain

In this tool chain we have four use cases Zip / contract, Zip / extract, Diff / compare, and Ls / compare. Each use case has its own set of inputs and outputs, e.g. Zip / contract takes a "File Tree" as input and delivers an "Archive" as output. Since the "Archive" contains product files the use cases Zip / contract and Zip / extract have tool impact (TI2) as they might inject faults into the product.

These use cases need to be analyzed for potential tool failures, e.g. "File Content Corrupted" in use-case Zip / contract and appropriate checks for these failures need to be assigned if possible, e.g. "Diff File Trees" in use-case Diff / compare. Note that in this tool chain the tools are not only sources for tool failures but can also act as sinks for tool failures by providing measures for failure detection or prevention. The effectiveness of these measures is expressed by the assigned TD level, which is omitted in the figure above.

3 Strategies for Potential Tool Failure Determination

This section defines terminology, goals and strategies for determining potential tool failures. In analogy to Laprie's fault/error/failure concept [6] and the ISO 26262 [1] vocabulary, we define the terms *tool fault*, *tool error* and *tool failure* as follows:

- *tool fault*: defect in tool code or design
- *tool error*: unexpected internal tool state at runtime, caused by tool fault or abnormal operating condition
- *tool failure*: unexpected tool output/result

We also distinguish between *concrete* and *potential* tool errors and -failures:

- *Concrete tool error/failure*: Specific tool error/failure, e.g. Zip v7.3 corrupts file contents with more than 4gb size in compression method "Ultra".
- *Potential tool error/failure*: Abstract tool error/failure, e.g. File Content Corruption.

The aim of tool evaluation and -qualification is to counter Murphy's law: that anything that can go wrong will go wrong. Tool evaluation requires the determination of the potential tool failures.

3.1 Goals for Potential Tool Failure Determination

The determination of potential tool failures should achieve various goals, which we introduce next. A desirable goal would be completeness in the following sense.

- *Completeness*: All concrete tool failures are subsumed by the determined potential tool failures.

Completeness is a very attractive goal, but it has the drawback that in practice it is hardly measurable as the number of concrete tool failures is usually unknown. Hence, if one does not use extreme abstractions for potential tool failures like the term "Tool Failure", which resembles the logical predicate "true" and covers the whole plane of possibilities, one can usually not be sure if all known and currently unknown concrete

tool failures are covered. What is measurable in practice is *relative completeness* of different strategies. One can apply various strategies and then compares the obtained potential tool failures with a previously disclosed list of concrete tool failures. One can count how many of these concrete tool failures are subsumed by the determined potential tool failures and use these ratios to compare the considered strategies.

A little less ambitious but still attractive are the following goals, which we will later use to judge the strategies introduced below:

- *Comprehensiveness*: No blind spots. All potential tool failures can be determined.
- *Uniformity*: All use cases are analyzed with the same method and same intensity.
- *Appropriate Abstraction*: The error descriptions are neither too vague nor too detailed.
- *Scalability*: The effort is acceptable even for large tool chains.

A determination strategy for potential tool failures is *comprehensive* if for every concrete tool failure it is able to determine a subsuming potential tool failure. In other words it is able to reach all concrete tool failures. If a strategy is not comprehensive the TCL might be inadequate.

A potential tool failure determination is *uniform* if the same methods and the same levels of rigor are applied to all use cases of all tools. Using unbalanced methods and levels of rigor is a typical sign for poor process quality.

The determined potential tool failures should also have an *appropriate* level of *abstraction*. If this level is too high no counter measures can be found and if it is too low unnecessary effort is introduced.

Finally the strategy should be *scalable* in the sense that the effort spent on tool evaluation should be acceptable and not grow drastically with the complexity of the analyzed tool chain determined by the number of tools, use cases, artifacts and data flow dependencies.

3.2 Different Views on Tool Failures

From an abstract point of view (see Fig. 3) the purpose for using a tool is to map input data, e.g. files, streams, etc., to output data. A tool failure leads to wrong outputs for valid inputs. An output is wrong if some parts of it do not map correctly to certain parts of the input.

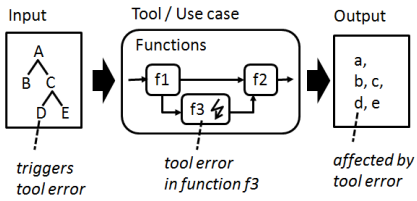


Fig. 3. Tool errors in functions affect artifacts

One the other hand tool failures are caused by errors in tool functions. For example an error occurring in a function f3 might lead to a wrong mapping of input part D to output part d. In order to describe a tool failure one can take two angles of view:

- describe what goes wrong inside the tool, e.g. error in function f3.
- describe what is wrong in the produced outputs, e.g. wrong part d in output file.

In the first description technique one refers to the internals of the tool, that is functions needed to accomplish the use case, whereas in the second one refers to properties or structure of the output data. Both descriptions may in addition refer to the properties of tool inputs that trigger the error. Note that both descriptions refer to the same tool failure. They mainly characterize this tool failure from different views.

These two views give rise for two tool failure determination strategies: Analyze what can *go* wrong in a tool or analyze what can *be* wrong with the artifacts.

The first strategy systematically refines the abstract error "Tool Failure" by analyzing the functions in the tool. We call this strategy *Function-based* failure determination. The second strategy only looks at the output data of tools and we call it thus *Artifact-based* failure determination. By going along the structure of output data one can systematically refine the abstract error "Artifact broken" into more concrete potential tool failures, e.g. output part d broken.

3.3 Function-based Strategy for Potential Tool Failure Determination

The function-based strategy analyses what can go wrong inside a tool and does this by decomposing the tool functionally. In this case functions can either be conceptual, e.g. sorting in a database, or in case the architecture or code of the tool are known also modeled from the internal structure of the tool.

Note that the same functions may take part in different use cases. There are also functions that are used by many tools for standard activities, e.g. "Iterating Files", and we call these *standard functions*. Standard functions characterize the tools and cause typical sets of potential tool errors. For examples in tools with the standard function "Iterating Files", typical tool errors are "File Lost" or "Unwanted File Added".

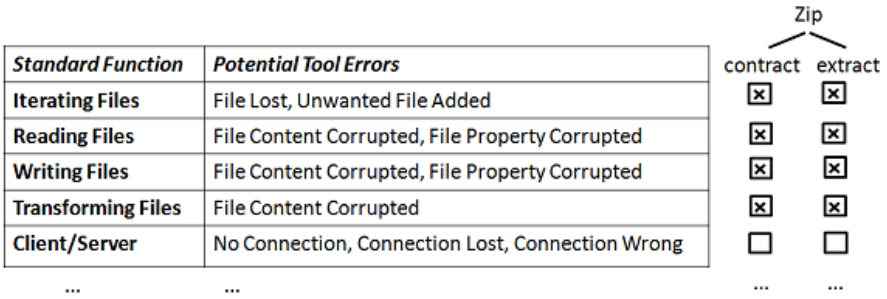
For each function or standard function we can associate a set of potential tool errors that may occur in tools having this function.

Once the sets of potential tool errors for functions and standard functions are defined, the function-based strategy essentially becomes a matter of selecting the appropriate functions or standard functions for each use case (see Fig. 4).

The potential tool errors for a use case are simply the union of the sets of potential tool errors of the functions selected for this use case. Sometimes similar tool errors are introduced from different functions.

Hence, after this selection phase the set of potential tool errors for a use case needs to be consolidated by subsuming similar tool errors. By now the function-based strategy has produced sets of potential tool *errors*, but the aim of tool evaluation is to determine potential tool *failures*. Only the externally observable effects of tool errors in terms of wrong artifacts being produced matter. To transform the set of potential

tool errors into a set of resulting tool failures one can apply an FMEA like inductive thinking. What can happen if this function produces this tool error? If one traces this question along the dependencies of functions one can derive corresponding potential tool failures.



Standard Function	*Potential Tool Errors*
Iterating Files	File Lost, Unwanted File Added
Reading Files	File Content Corrupted, File Property Corrupted
Writing Files	File Content Corrupted, File Property Corrupted
Transforming Files	File Content Corrupted
Client/Server	No Connection, Connection Lost, Connection Wrong
...	...

Fig. 4. Assigning standard functions to use cases

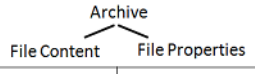
In the Zip example from the previous section we can decompose the use-case Zip / contract into the following functions: Iterating Files, Loading Files, Transforming Files, Writing Files. Each of these functions brings along its own set of typical potential tool errors, which can be consolidated (see Fig. 4) and then transformed to tool failures.

3.4 Artifact-based Strategy for Potential Tool Failure determination

The artifact-based strategy identifies potential tool errors by decomposing the structure of the artifacts and looking for things that may break or get flawed. To do this systematically we can employ the guide word confrontation technique known from the HAZOP analysis.

In this technique one creates a matrix where the columns are labeled with the things that may be faulty, that is the artifacts or their parts/properties, and the lines are labeled with guide words that describe certain kinds of faults, e.g. “Too many”, “Too few” or “Wrong” (correct amount, but wrong content).

For every guide word - artifact pair one starts thinking if this combination is meaningful and if so what typical potential tool failures might be associated with this combination. The resulting potential tool failures are then written into the corresponding cell of the matrix (see Fig. 5).



Too few	File Lost	n.a.
Too many	Unwanted File Added	n.a.
Wrong	File Content Corrupted	File Property Corrupted

Fig. 5. guide word – artifact confrontation

Sometimes the potential tool failures that come out of such an analysis are too coarse. A way out is often to further decompose the artifacts by their structure or some other properties and then to confront each part/property with the guide words again.

In our case we can decompose the artifact "Archive" into the parts "File Content" and "File Properties". By doing this we do not end up with the potential tool failure "File Corrupted", but with two finer potential tool failures "File Content Corrupted" and "File Properties Corrupted", which can now be detected by different measures, e.g. "Diff File Trees" and "Compare ls -l" (see Fig. 2).

3.5 Do these Strategies Achieve the Goals?

Above we have stated *comprehensiveness*, *uniformity*, *adequate error abstractions* and *scalability* as goals for potential tool failure determination. As every tool failure that is of interest must have a cause *inside* the tool and an effect on some artifact *outside* the tool, both strategies can principally identify every potential tool failure. Hence, both strategies are comprehensive.

Since both strategies use systematic confrontation techniques (standard function to use-cases, guide word to artifact) they can also be regarded as uniform.

According to our practical experience the abstractions of the identified potential tool failures are often inadequate in both strategies. Sometimes the descriptions are too coarse, e.g. "File Fault", and one cannot assign an appropriate detection or prevention measure. Sometimes the descriptions are unnecessarily detailed and one ends up with many slightly different tool failure descriptions that all can be handled by the same detection or prevention measure. In our tool chain example the function-based strategy in practice tends to give us many versions of the failure "File Content Corrupted" depending on the function that has failed, e.g. "File Content Corrupted due to false reading". In the end it does not matter if the corruption happens while reading, transforming, or writing files as all these corruptions can be detected by the same check (Diff File Trees).

Nevertheless there is a way to deal with inadequate abstractions. If appropriate measures for detection or prevention are in sight, but the current failure descriptions are too coarse for them one can refine the decomposition of functions or artifacts and re-apply the identification strategy for potential tool failures.

If the tool failure descriptions are too detailed one can subsume a multitude of detailed descriptions with one more abstract tool failure description. If we include the decomposition refinement and the failure consolidation as a post processing step for both strategies, we can achieve the goal of having failure descriptions with an adequate level of abstraction.

What about scalability? In theory both strategies should scale well for large tool chains as they only analyze isolated pieces of a tool chain, one tool/use-case or one artifact at a time. In practice we observed the function-based strategy to cause significantly more effort. One reason was tool error consolidation. We typically had to assign many standard functions to use cases and ended up with large sets of similar potential errors that had to be subsumed. A second reason was the difficulty to trans-

form tool errors to tool failures. The internal dependencies between functions are often not known to the public. If so, we have to think pessimistically and assume that every tool error will corrupt all output artifacts. Even the log files which often provide a chance to detect tool errors might be affected. Finding effective counter measures often requires an analysis of the artifacts as it is done in the artifact-based strategy right away. On the other hand, if sufficient details on a tools internals are available, a rigorous function-based strategy pays off when it turns out that a tool has TCL2 or TCL3 and requires qualification. Knowing the hardly detectable internal tool errors and critical functions is very valuable for writing a tailored qualification plan or building a qualification kit.

4 Tool Support for Tool Evaluation: Tool Chain Analyzer

ISO 26262 requires all tools used in development to be analyzed. Tool chains in industrial practice are rather large and may lead to many potential errors to manage. Assigning detection or prevention measures to all these errors, subsuming errors and maintaining their relation to use cases, standard functions and artifacts is a complex task, which needs tool support. Within the research project RECOMP Validas has built such a tool called Tool Chain Analyzer (TCA).

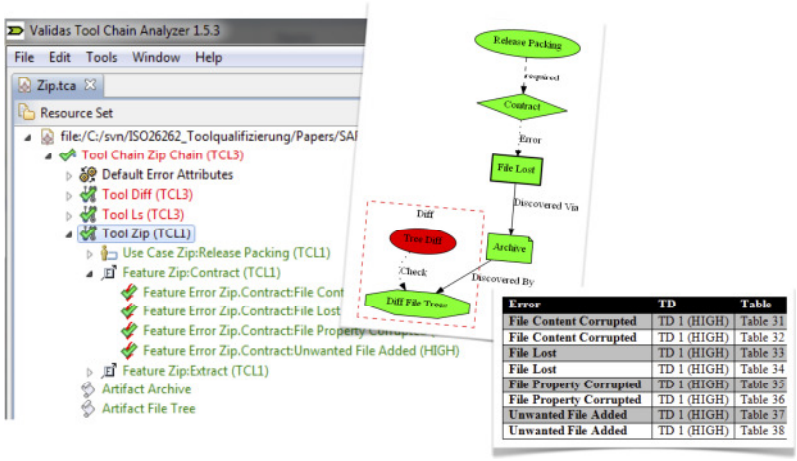


Fig. 6. Tool Chain Analyzer with generated figures and tables (MS Word)

The TCA (see Fig. 6 and Fig. 7) allows to model a tool chain structure in terms of tools having use cases which are reading or writing artifacts. The TCA also allows to model the confidence in a tool chain in terms of tool failures, checks and restrictions as shown above (see Fig. 2) together with their TD. From the tool chain model and confidence model the TCA can automatically computes the TCL of all tools in the following way: First, the TCA obtains the TD for each potential failure by taking the TD with the lowest number (highest probability) the user has assigned for one of the assigned counter measures.

Second, the TCA computes the TD for a use case by taking the worst TD for any potential failure identified for this use case. Third, by combining the TD for a use case with the TI of this use case according to the ISO 26262 table the TCA derives a TCL for this use case. Finally, the TCL of a tool is the worst TCL for any use case.

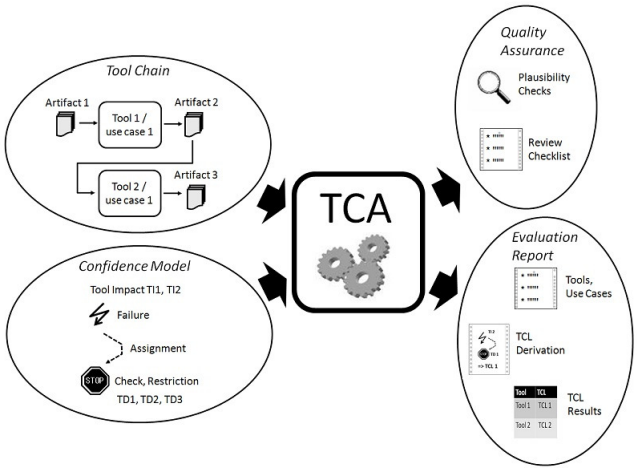


Fig. 7. Inputs and outputs of the TCS

The TCA also offers lots of plausibility checks for the tool chain and confidence model. For example if an detection measure from Tool B is assigned to a potential failure of tool A then there must be a data flow in terms of input/output artifacts from tool A to tool B, otherwise the assignment of this detection measure is invalid.

The TCA can also generate a MS Word report, which contains detailed tables and figures for each identified potential tool failure, such that the computed TCL becomes plausible, comprehensible and checkable by review. The structure of this word report is designed such that it can be directly used as a part for the tool criteria evaluation report required by ISO 26262.

5 Conclusion

Tool evaluation is a critical step that comes before tool qualification. Besides determining the TCL the tool evaluation often identifies ways to rearrange or extend the existing work flow such that tool qualification becomes obsolete.

We have applied both the function-based and the artifact-based identification strategies for potential tool failures in a large scale industrial project using the TCA.

Our experience is that the function-based strategy tends to yield failure descriptions that are too detailed or overlapping. While having very detailed tool failure descriptions is useful for tool qualification, it is unnecessary for tool evaluation, which

is only concerned with TCL determination. We experienced the artifact-based strategy to produce failure descriptions with more adequate levels of abstractions right away. In contrast to a tools internals the structure of artifacts is often known and allows to refine failure descriptions on demand.

Our experience with the tool TCA is that it greatly helps to improve the quality of the obtained evaluation report. In particular the support it offers in form of plausibility checks, review assistance and refactoring helps to iteratively develop a comprehensive and consistent model of the tool chain.

However, both the strategies and the tool have potential for further research and improvements. For the strategies further experience is required, e.g. measuring relative completeness and the reproducibility of results by letting multiple people with different backgrounds apply the strategies.

Also the TCA could be improved, e.g. by establishing reusable catalogues for standard functions, patterns for functional decompositions for various kinds of tools or appropriate structural decompositions for various kinds of artifacts, e.g. source code files or object code files. Nevertheless the current TCA forms a good platform for analyzing tool chains and confidence models and to try out various approaches.

Acknowledgment

This work has been supported by ARTEMIS and the German Federal Ministry of Research and Education (BMBF) within project RECOMP under research grant 1IS10001A.

References

1. International Organization for Standardization: ISO 26262 Road Vehicles – Functional safety–. 1st Edition, 2011-11-15
2. Stürmer I. and Conrad M.: Code Generator Certification: a Testsuite-Oriented Approach. In: Proceedings of Automotive-Safety & Security, 2004.
3. Schneider S., Slotosch O.: A Validation Suite for Model-based Development Tools. In: Proceedings of the 10th International Conference on Quality Engineering in Software Technology, CONQUEST 2007.
4. Beine M.: A Model-Based Reference Workflow for the Development of Safety-Critical Software. In: Embedded Real Time Software and Systems, 2010.
5. Hillebrand J., Reichenpfader P., Mandic I., Siegl H., and Peer C.: Establishing Confidence in the Usage of Software Tools in Context of ISO 26262. In: Flammini F., Iologna V., Vittorini V. (eds.) SAFECOMP 2011. LNCS, vol. 6894. Springer, Heidelberg (2011)
6. Laprie J.C. (ed.): Dependability: Basic Concepts and Terminology. Book. Springer-Verlag, 1992. ISBN 0-387-82296-8