

# Application Design Assessment Engines

Hans H. Kron

*Independent Safety Assessor for Railway Signalling & Maglev Operation Control Systems, Am Mandelberg 1, D-76831 Birkweiler, Germany, dr.kron@t-online.de*

**ABSTRACT:** A characteristic feature of maglev operation control systems is the need to design each installation to meet the individual requirements for a specific application and its topography. For efficiency, an operation control system is usually split into a generic (application-independent) part and a set of application data for each specific application. This paper introduces methods for the automatic verification of application data as part of the safety assessment of the data-driven control system.

## 1 DATA-DRIVEN SYSTEMS

### 1.1 Three Layers of an Operation Control System

“The operation control system (OCS) of the Transrapid monitors and controls the various subsystems, integrating them to form a safe, automated overall system” (Schünemann 2003). Like other train control and protection systems, OCS has a modular architecture which can be viewed as a three-layered structure, consisting of “generic products” and “generic applications” to form the unique “specific application” for a particular installation or a particular vehicle. Figure 1 shows an example taken from EN 50129 Figure 9 (CENELEC 2002).

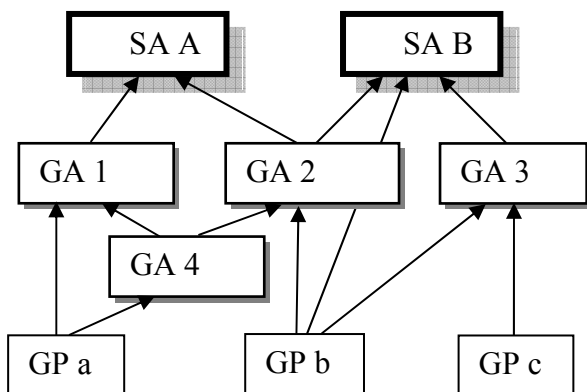


Figure 1: Dependencies between generic products (GP), generic applications (GA), and specific applications (SA).

A “Generic Product” can be re-used for different independent applications. For instance, a hardware platform - like the “SIMIS” vital computer system used in several distinct OCS subsystems - is a generic product (Materne & Seib 1997).

A “Generic Application” can be re-used for a class of applications with common functions. A generic application may use generic products. An example would be the Transrapid “decentralized safety computer” DSC which makes use of the generic product “SIMIS” and can be used in different decentralized control sections of maglev lines. Another generic application is the vehicle safety computer VSC to be used for a class of maglev vehicles.

A “Specific Application” is the unique system designed for one particular installation, i.e. an individual control section or vehicle. Thus, a specific application has only one instance, e.g. the DSC 1 for Long Yang Road Station on the Shanghai SMT main line. It may use generic products and applications.

Obviously, a specific application is more than the sum of its generic parts. The difference lies in the set of application data representing the application design.

### 1.2 Systems configured by application data

To quote EN 50128: “A characteristic feature of railway control and protection systems is the need to design each installation to meet the individual requirements for a specific application. A system configured by application data allows type-approved generic software to be used, with the individual requirements for each installation defined as data (application specific data). This data is normally in the form of tabular information or an application specific language which is interpreted by the generic software.” Figure 2 shows how the generic software interacts with its environment, thereby using or in-

interpreting the application data. Generic software, static application data, and dynamic application data (variables) together form a “data-driven” or “table-driven” system. In this model, static application data will be generated before commissioning, best placed in some read-only memory, and protected by static check sums (e.g. CRC, MD4 or MD5 signatures).

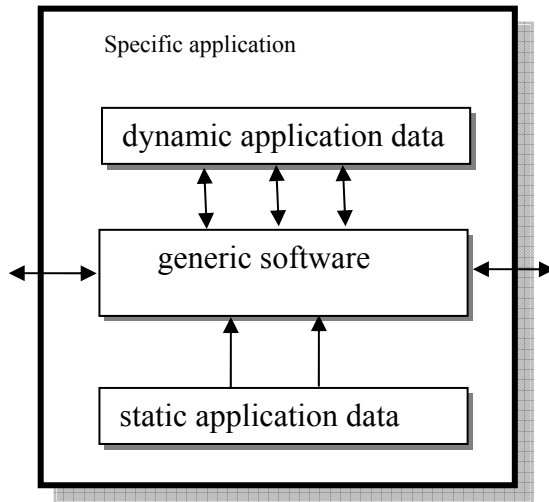


Figure 2: A specific application of a data-driven system.

### 1.3 Static application data

Other names for static application data are “customized”, “installation-specific” or “configuration” data. The types and purposes of application data are varied:

- hardware parameters, port addresses,
- physical and logical communication parameters,
- command tables, message distribution tables,
- element attributes (see below),
- topographical data,
- route tables or route maps,
- speed, acceleration, and jerk limits,
- speed profiles, braking and levitation curves,
- parameters for hardware safety monitoring and safety reactions (see below) and
- parameters for built-in self tests (see below).

By “elements” we mean the basic objects of the application domain, for OCS e.g.

- guideway sections (tracks or switches),
- location references,
- slope elements,
- velocity elements,
- operational destination points,
- stopping points,
- stations (door release),
- maintenance areas,
- power rails,
- propulsion elements,
- radio areas and
- vehicles.

Each of these elements has its own set of attributes, depending on its type. Element attributes might be:

- name of element (identifier, uncoded text),
- type and subtype of equipment,
- physical location (wayside and indoor),
- safety distances, clearance points,
- durations (e.g. switch throwing times),
- general properties (e.g. number of guideway switch positions),
- logical properties (application conditions),
- selection of features and special functions,
- references to neighbouring elements, and
- references to dependent or interlocked elements.

Parameters for hardware safety monitoring and safety reactions might deal with:

- form of input and output redundancy (antivalent or equivalent digital signals),
- tolerable duration of input discrepancies,
- tolerable duration of transmission failures and
- sets of elements in one subsystem which have to enter a fail-safe state if a neighbouring subsystem or a subordinate component fails.

Parameters for built-in self tests might deal with:

- elements or equipment to be tested regularly,
- resources (like memory areas) to be tested,
- self-test activation time or frequency.

### 1.4 Application data preparation

The range of applications (the “variability”) of the data-driven system must be defined. For each element type, the number and type of attributes must be defined. In conclusion, there must be an “data preparation handbook” supporting plant application engineering, and in particular application data preparation. The data preparation handbook must be reviewed during generic validation and assessment. Together with the generic system specification, the handbook serves as a bridge between the generic system and the design of the specific application.

## 2 SAFETY OF APPLICATION DESIGN

### 2.1 Basic requirements & considerations

Quoting EN 50128 (CENELEC 2001) again: “For a safety critical system, the high level of resources required to develop software to achieve the required system safety integrity level ... makes the adoption of a system configured by application data very attractive because it allows the re-use of generic software. However, as the safe operation of the system is likely to depend on the correctness of the data, the procedures used for development of the data must

also be to an appropriate system safety integrity level.”

## 2.2 Hazard identification

In principle, any incorrect information contained in the application data might be hazardous. Even a single incorrect bit - e.g. of a local speed limit - can have catastrophic results, as demonstrated by some recent railway accidents and near-misses. So one lesson we have to learn is that in checking application data, spot checks and random samples are not sufficient. Data tests and data analysis must be as systematic and complete as possible, at least for safety-relevant data.

It is, however, not always easy to decide if a particular piece of application data is safety-relevant or not. With modern object-oriented generic software, the possible consequences of a data error are not easily analysed. Some trivial application data errors prove to be as “treacherous” as generic software coding errors, leading to a severe, hazardous fault only after a rare and intricate combination of events and operations.

It is wrong to assume that all application data errors will be detected during dynamic testing of the system. One class of errors undetected by the usual integration tests are incomplete or missing activations of built-in self tests. The same holds for missing or wrong parameters of safety-relevant inputs.

Some data-driven functions are very hard or time-consuming to test. An example are the detailed safety integrity functions handling hardware or transmission failures, input discrepancies, operator’s command syntax errors, etc. An exhaustive test of all failure modes for all hardware components, interfaces and operator inputs is impossible, so the correctness of the application data for these safety integrity functions has to be shown by analysis rather than test.

Moreover, there is Dijkstra’s famous corollary “Program testing can be used to show the presence of bugs, but never to show their absence” (Dijkstra 1972). It is generally recognized that testing alone cannot prove the correctness of programmable systems. Even for a simple program with a few lines of code, an exhaustive path test would take much more time than the age of the universe. This does not only apply to the “positive” tests demonstrating that the required system functions will perform correctly when invoked correctly under normal circumstances. Even more so, it applies to “negative” tests demonstrating that incorrect commands, unusual operations, rare combinations or sequences of events, and odd circumstances will never produce an unsafe state.

In particular, dynamic testing against the requirement specification can never convince us that the system does not contain unexpected and unwanted functions, elements, or features. In practice, it happens quite often that a data-driven system contains

- imprecisely or incompletely documented functions unknown to validator and assessor,
- provisional, obsolete, or superfluous elements not removed completely from application data,
- features not required by the specific application, but supported by the generic system and inadvertently activated by application data, and
- inadvertent side effects between unrelated elements.

These errors or anomalies can only be caught by systematic analysis of the application data, not by any reasonable amount of testing. Figure 3 indicates that there are large differences both

- in the visibility of application data errors during the regular use of the OCS, and
- in dynamic test coverage (the number of dynamic test cases compared with the size of the input space).

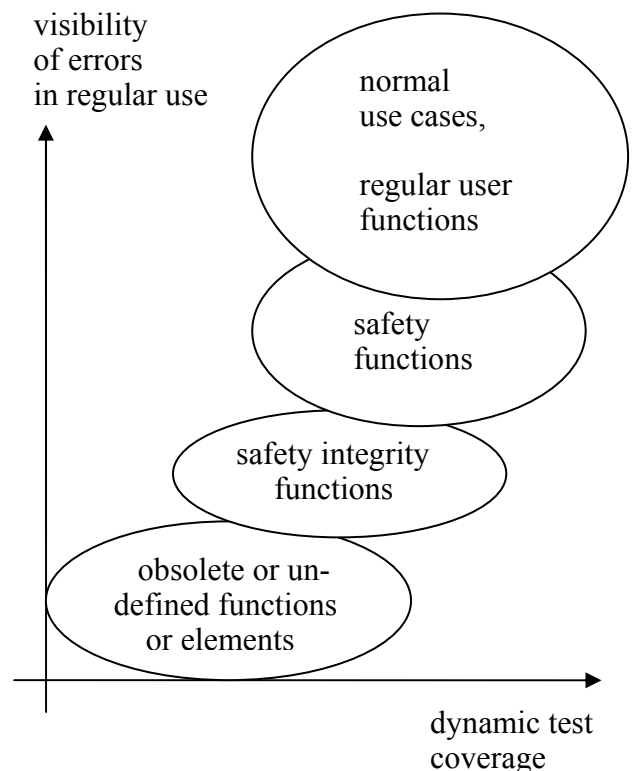


Figure 3: Visibility and test coverage of systematic errors in application data.

## 2.3 Consequences for the safety life cycle

An OCS may only be approved by the safety authority and accepted by the maglev transport authority if

both the generic subsystem and the corresponding application data have successfully passed the safety life cycle, including

1. verification - to determine by analysis and test that the output of each life cycle phase meets the requirements of the previous phase,
2. validation - to demonstrate by analysis and test that the system meets in all respects its specified requirements, and
3. safety assessment (see chapter 3).

EN 50128 calls for a data generation plan, a data test plan, and a data test report. The data test activities of the supplier should produce a statement regarding the data coverage of the data-driven vital systems, on the analogy of the control flow coverage achieved by program module testing.

For the life cycle phase “modification and retrofit”, change control procedures have to be established. Complex changes can be reduced to three elementary scenarios:

1. There are changes in application design, but not in the functionality of the generic software. The new software release contains the original generic software and a revised set of application data.
2. The generic software has to be changed, but it is still compatible with the original application data. The new software release contains the revised generic software and the original set of application data.
3. The generic software has to be changed together with the interface between generic software and application data. The data preparation handbook has to be revised. The new software release contains the revised generic software and a revised set of application data.

### 3 SPECIFIC APPLICATION ASSESSMENT

#### 3.1 Scope and definition

The safety case for a specific application shall be divided into two portions (CENELEC 2002):

- the “application design safety case” containing the safety evidence for the theoretical design and
- the “physical implementation safety case” containing the safety evidence for the physical implementation.

Accordingly, there shall be two safety assessment reports, for application design and for physical implementation respectively. Safety assessment in general

- must determine whether verification and validation have achieved a system that meets the specified requirements,

- form a judgement as to whether the system is fit for its intended purpose, and
- determine if the subsystems are of the defined safety integrity level (SIL).

In the remainder of this paper, we shall deal with application design assessment only.

#### 3.2 Steps of application design assessment

We assume that the system requirements specification is split into a “generic system requirements specification” and a “plant specification” (CENELEC 2001). A more general term for “plant specification” is “application requirements specification”. The plant specification contains the specific requirements of the particular installation (e.g. track layout, locations, route tables, speed limits and profiles), and refers to the applicable design standards (e.g. rule books, signalling safety principles and guidelines).

Figure 4 shows the steps of a specific application assessment which are relevant to our topic, viz application design assessment. For simplicity, Figure 4 does not distinguish between hardware and software; to be complete, it would have to include hardware configuration documents, too.

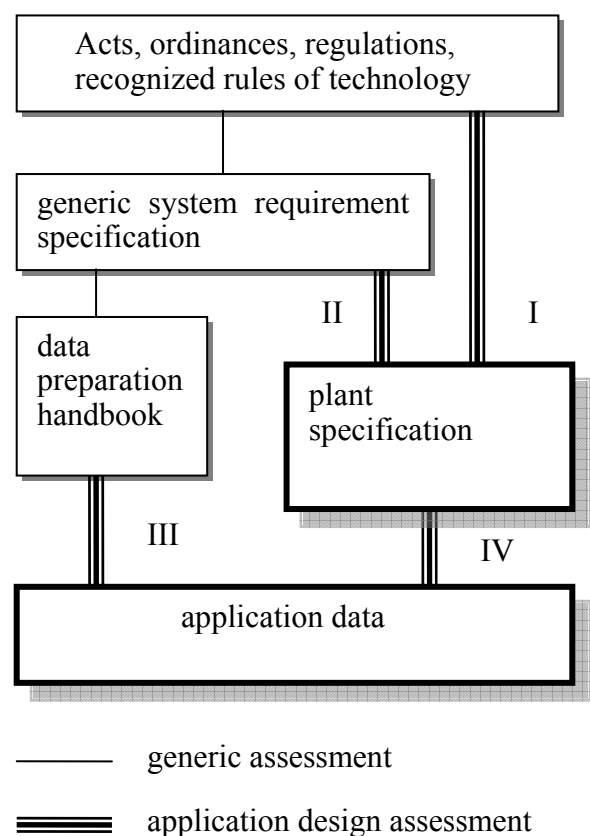


Figure 4: Application design assessment.

Four basic steps of application design assessment are represented by thick lines:

- I. checking that the plant specification is complying with acts, ordinances, regulations, and recognized rules of technology,
- II. checking that the plant specification is compatible with the generic specification,
- III. checking that the application data conform to the generic data structures and the data preparation handbook, and
- IV. checking against the plant specification that the application data are complete and correct.

## 4 ASSESSMENT ENGINES

### 4.1 Goals and objectives

The theoretical assessment of the application design and the set of application data is a cumbersome yet safety-critical and mandatory task. IEC 61508-3 Ed.2 Annex H on data-driven systems is warning us: “Reliance upon the human being is likely to introduce errors, so automation should be seriously considered; ... so provide tools and aids to assist in creation of correct data” (IEC 2005). The careful manual check of a set of application data with average complexity may easily take some weeks (Holzmüller et al. 2006).

The central idea presented in the following sections is to develop automatic tools for the specific application assessment of a maglev OCS, called “application design assessment engines”, in order to

1. comply with modern safety-related software standards,
2. provide full documentation and repeatability of the assessment steps,
3. ensure the compatibility of generic programs and specific application data,
4. capture application design errors early,
5. increase the completeness and efficiency of detecting application data errors,
6. reduce the amount of site acceptance tests necessary,
7. provide complete and quick re-assessment in case of application design modifications, rather than rely solely on difference analysis and regression tests, and to
8. support the modification or extension of maglev lines without substantial interruptions of commercial service.

### 4.2 Techniques

The following techniques are not confined to automatic assessment. The basic principles can also be applied to data verification or validation methods, either manual or automatic.

#### 4.2.1 Formal verification

This technique consists of reading both the plant specification and the set of application data, and verifying that they are semantically equivalent. The application data may be represented as object code, assembly language modules, or in some domain-specific intermediate language. The equivalence check must cope with the differences in language, representation, data structure, level of detail, naming, order, rounding, and comments.

One variant of the technique requires the assessor to independently define a formal plant specification (see Figure 5). This variant is mandatory if there is no formal representation of the plant specification at all, or if the plant specification is not readable by the assessment engine. The variant is an option if the assessor does not want to make use of an existing formal specification derived by others. The variant has been used extensively for the application data assessment of electronic interlockings (Kron 1999).

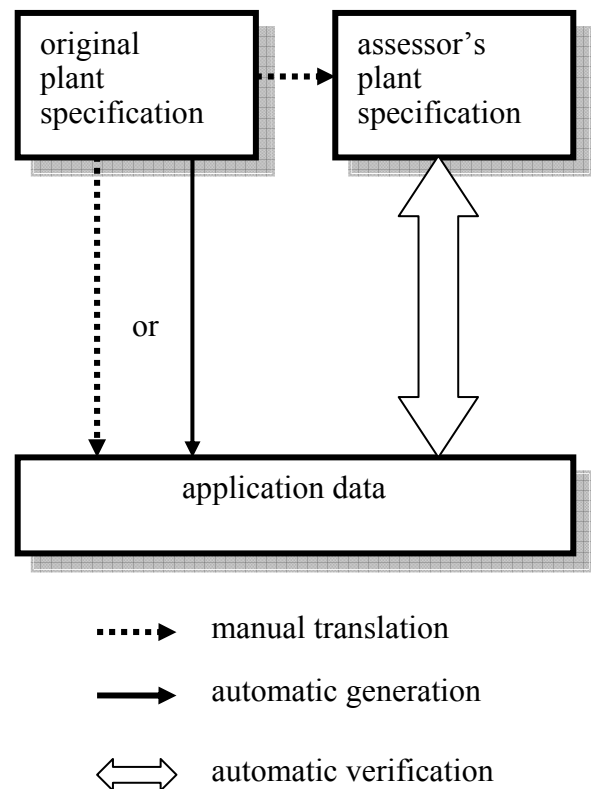


Figure 5: Formal verification, second variant.

#### 4.2.2 Diverse forward translation

This technique generates, independently of the original data preparation process of the design team, a second set of application data from the plant specification. Forward translation is successful if the original and the diversely compiled set of application data are equivalent (see Figure 6). Here, the equivalence check only needs to cope with differences in naming, order, rounding, and comments. In every other respect, the data structures must be equal.

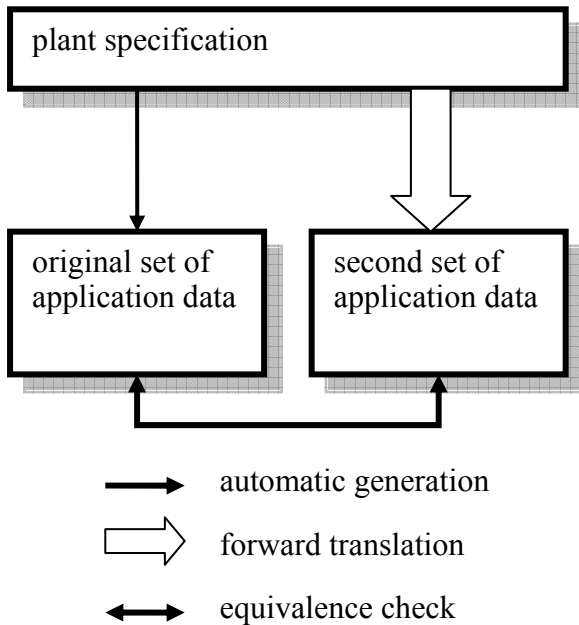


Figure 6: Diverse forward translation.

This technique has been used for the track data validation of LZB L72 CE automatic train control systems (Holzmüller et al. 2006). A diverse track data compiler reads the common formal representation of the plant specification and generates a second set of application data in assembly language. This set is then compared automatically with the original set.

#### 4.2.3 Diverse backward translation

This technique (Figure 7) consists of reading the application data (as object code, assembly code, or intermediate code), disassembling, and decompiling the data to regain the plant specification or an essential part thereof. Backward translation is successful if the original and the decompiled plant specification are equivalent.

#### 4.2.4 Conformity checks

This technique checks that the set of application data conforms to the generic data structures, the data preparation handbook, design standards, rules of technology, etc. The assessment engine contains a set of rules for the detailed checks. The set of rules may be expanded incrementally.

#### 4.2.5 Assertion checks

Some safety requirements and plausibility checks may be formally represented as predicate logic conditions - so-called assertions - on the set of application data. For a given set of application data, the assessment machine checks if all assertions are true. The set of assertions may be expanded incrementally.

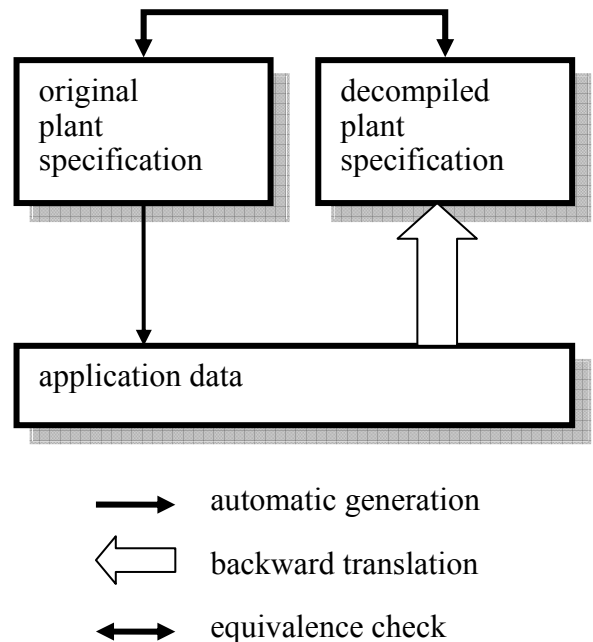


Figure 7: Diverse backward translation.

#### 4.2.6 Pattern matching or recognition

Usually, a specific application contains many elements of the same type (e.g. guideway sections or switches) with large numbers of individual attributes. The combinatorial complexity, however, is limited. Each type is likely to have only a few subtypes (e.g. 2-way-switches and 3-way-switches) exhibiting fixed patterns of attributes. One variant of the technique is to define manually some attribute patterns (i.e. element subtypes) and to have the assessment engine to classify all elements accordingly. Then, the assessor can check if each element has been classified as expected. There might be a few elements that do not match any defined pattern; these must be checked individually. A future variant of the technique would try to find the patterns automatically.

#### 4.2.7 Checklist generation

A list generator supports the manual analysis of application design and the site acceptance tests by generating highly structured and individualized checklists, e.g. for checking all elements of a particular type, or for checking all available routes and their interlocking conditions. It is safety-relevant to generate the checklists not from the application data, but from the plant specification (or the assessor's plant specification).

## 5 IMPLEMENTATION ISSUES

### 5.1 Languages

For constructing and maintaining assessment engines, we need high level programming languages with high productivity and an affinity to semantic

nets, knowledge bases, rule-based logical inference, pattern recognition, and symbol manipulation. On the other hand, real-time capabilities are not needed. Some distinguished candidates are Ada, Haskell, and Prolog.

### 5.1.1 Prolog

Prolog is a descriptive programming language, describing facts and formal relationships between objects. It is based on predicate calculus. In the area of assessment engines, Prolog has been used for the automatic assessment of the application data of electronic interlockings (Kron 1999), and for the automatic validation of route maps of ETCS Radio Block Centers (Eisen & Voigt 2003).

### 5.1.2 Haskell

Haskell is a functional, strongly typed programming language based on lambda calculus. It offers substantially increased programmer productivity, concise and maintainable code, and fewer errors. Haskell programs are easy to understand. The diverse track data compiler for the LZB L72 CE automatic train control system was successfully and efficiently written in Haskell (Holzmüller et al. 2006).

### 5.1.3 Ada

Ada is a procedural, object-oriented, strongly typed programming language for large-scale software systems. Ada's goals are high reliability and dependability for safety-critical environments. Emphasis is on program readability and maintainability. The use of Ada subsets is highly recommended by IEC 61508-7 (IEC 2000), DO-178B and EN 50128 (CENELEC 2001). There is an ISO/IEC guideline for the use of Ada in high integrity systems, and validated Ada compilers are available.

## 5.2 Validation of assessment engines

Application design assessment engines have to be validated. There must be documented test cases to demonstrate that the tool works as intended (IEC 2005), in particular that it correctly detects errors in the application data. The test cases can be easily derived by mutating correct sets of application data. This technique is also known as "error seeding".

## 6 CONCLUSIONS

Although the safety assessment of a specific application will usually be based on the results of verification and validation, the assessor has to form an independent and comprehensive judgement as to whether the system is fit for its intended purpose and whether its subsystems are of the required safety integrity

level. Firstly, by creating an assessment engine, the assessor will deepen his or her knowledge of the control system's functionality. Secondly, by using the assessment engine for a specific application, the assessor will deepen his or her knowledge of the plant specification and the details of the technical solution.

Seen from another angle, assessment engines help us to counterbalance the usual bias on testing as opposed to theoretical analysis. Testing can never prove correctness, so at least for the limited area of application data, we should do better and formally prove them correct.

Another important message is that the techniques used by designers, verifiers, validators, and safety assessors should be as diverse as possible.

## REFERENCES

CENELEC 2001. *European Standard EN 50128, Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems*. Brussels: CENELEC.

CENELEC 2002. *European Standard EN 50129, Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling*. Brussels: CENELEC.

Dijkstra, E. W. 1972. Notes on Structured Programming. In O.-J. Dahl, E. W. Dijkstra & C. A. R. Hoare, *Structured Programming: 1-82*. London, New York: Academic Press.

Eisen, J. & Voigt, E. 2003. XML-basierte Prüfung der RBC-Streckenprojektierung. *Signal + Draht (Rail Signalling and Telecommunication)* 95 (5 / 2003): 18 - 22.

Holzmüller, B., Hofmann, T. & Renninger, G. 2006. Validierung des Bereichsdatencompilers (BDC) für die LZB L72 CE. *Signal + Draht (Rail Signalling and Telecommunication)* 98 (3 / 2006): 25 - 30.

IEC 2000. *International Standard IEC 61508-7 Ed.1, Functional safety of electrical / electronic / programmable electronic safety related systems - Part 7: Overview of techniques and measures*. Geneva: IEC.

IEC 2005. *International Standard IEC 61508-3 Ed.2 Committee Draft (CD), Functional safety of electrical / electronic / programmable electronic safety related systems - Part 3: Software requirements*. Geneva: IEC.

Kron, H.H. 1999. Automatische Verifikation der Projektierungsdaten von ESTW. *Signal + Draht (Rail Signalling and Telecommunication)* 91 (12 / 1999): 30 - 31.

Materne, R.-T. & Seib, G. 1997. Die fahrwegseitige Sicherung und Steuerung des Transrapid. *Signal + Draht (Rail Signalling and Telecommunication)* 89 (7-8 / 1997): 23 - 27.

Schünemann, F. 2003. Die Betriebsleittechnik des Transrapid. *ZEVrail Glasers Annalen - Sonderheft Transrapid 2003*: 88 - 94.