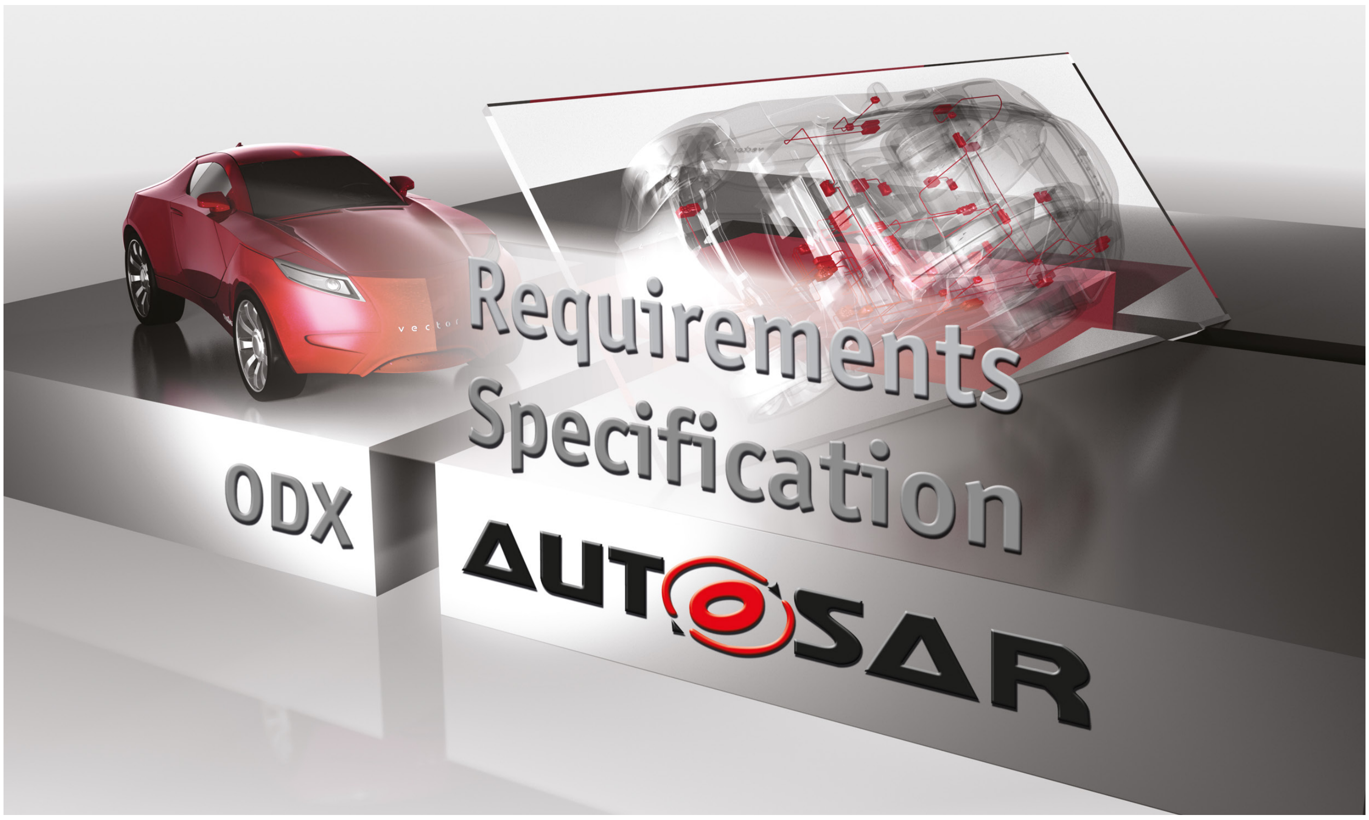
From Diagnostic Requirements to Communication

Standardization is the Trend in the Development of Automotive Electronics



A key aim of open architectures, configurable components and harmonized exchange formats is to let developers focus more on the development and reuse of innovative and product differentiating functions. In recent years, a number of independent standards have been created, all of which have affected processes and tools for diagnostic development – in particular ODX and AUTOSAR. At the same time, the systematic capture, management and tracking of requirements took hold, which also had a significant impact on processes, methods and tools.
Is it possible to do without one or more standards? Is there a super-standard? Or can the standards and methods be combined with one another effectively and efficiently?

Requirements Engineering

The development of a system starts with the requirements from the user's perspective. The capture of requirements marks the beginning of an iterative process **(Figure 1)**, in which requirements of a system are progressively made more specific and precise. If the solution space for fulfilling requirements is still large, the later specification describes individual subsystems precisely and without ambiguities.

In practice, requirements differ in terms of how specific and precise they are. Text-based requirements describe a system property to be fulfilled in text form, usually incompletely and purposely fuzzy, phrased or just in note form. Specification requirements, on the other hand, are precise and not only describe the requirement itself, rather they also include the solution and leave very little freedom for the specification. Formal languages are often used for the description, which are appended to text-based requirements in files. Reference requirements contain a reference to a specification, e.g. "as in the previous system". Technically, these reference requirements actually reference specifications in other databases or data management systems in many cases.

Ideally, requirements are defined as precisely as possible from the start, but only as specifically as necessary. Unclear or ambiguous requirements lead to considerably increased effort over the course of the development process, because clarification means there is a need for additional coordination, and it often results in a specification change. In the least favorable case, the system implementation may even need to be modified. On the other hand, requirements that are unnecessarily specific can often actually obstruct the path to the quickest and most cost effective solution. If aspects of a solution path are intermixed with requirements early on, this unnecessarily reduces the solution space. Often, this also eliminates the opportunity for re-use. Especially when requirements change over the course of development, it is important to separate the substantial requirements from relicts of earlier solution approaches.

During development, the totality of implementation progress for all requirements offers a good overview of the implementation progress of the total system or of a subsystem (maturity level tracking).

If you want to systematically exploit the advantages of a requirements-driven process, then the process described above must be applied to all subsystems, including those of different development disciplines that are actually independent. Naturally, this also applies to diagnostics.

Today, spreadsheet-oriented tools and databases are usually used to manage requirements. Here, requirements are either not described formally, or they are only described formally in part. These tools must be flexible enough to capture and track all requirements – even those that are very fuzzy.

Regarding the specification, various other tools have become established in the various disciplines, e.g. modeling and authoring tools, which usually generate a formal specification. In contrast to user requirements, precise definition of the content is the primary goal and not flexibility, and this fundamental difference results in different, specialized tools. Consequently, classic requirement management tools can only be used meaningfully up to a certain level of detail. This also applies to diagnostics.

Standardized exchange formats are specially designed for a specific discipline. ODX, for example, specifies data that is relevant to the diagnostic tester. Exchange formats usually use a formal data model that assures a consistent specification that is complete in its details. On the other hand, these formats are too restrictive for formulating fuzzy requirements. Classic requirement management tools are well-suited to describing text-based diagnostic requirements. The standardized data exchange format ODX, meanwhile, would be unsuitable for describing or exchanging these text-based requirements, because it is too formal and precise.

ECU Software

Today, AUTOSAR (AUTomotive Open System ARchitecture) is the reference architecture for ECU software in the automotive industry. AUTOSAR standardizes the description of individual component or vehicle functions and the description of the overall system.

The diagnostic software in AUTOSAR consists of the three basic software modules DCM, DEM and FIM.

The DCM (Diagnostic Communication Manager) implements diagnostic communications according to UDS and OBDII. The DEM (Diagnostic Event Manager) implements a fault memory and manages fault status and supplemental information on fault symptoms. In the case of active faults, the FIM (Function Inhibition Manager) prevents execution of certain functions and suppresses secondary errors.

DCM, DEM and FIM are configured by the ECU Configuration Description (ECUC). Their contents are best understood by illustrating how requirements relate to the configuration of software components.

Fuzziness and flexibility, which are advantageous in capturing requirements, must be avoided in configuring the ECU software. The software must be described precisely and unambiguously for all operating conditions that occur.

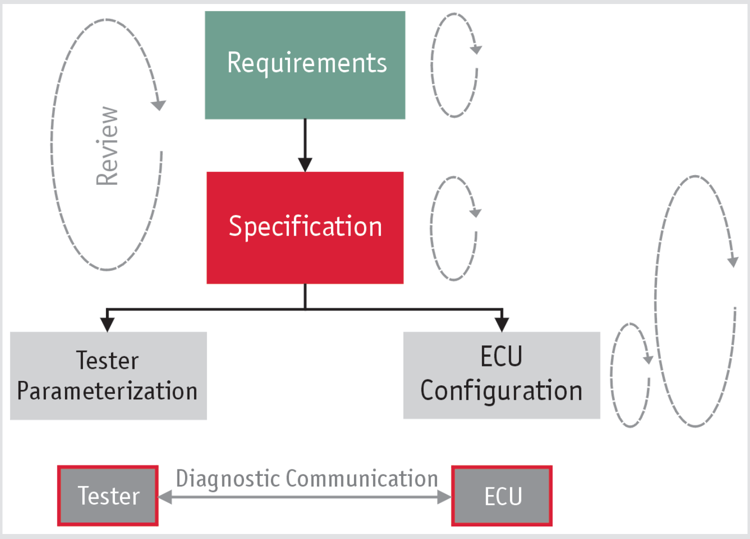


Figure 1:
Iterative development process

Significant contents of the diagnostic data that are relevant to the software configuration include the diagnostic services that can be called by an external diagnostic tester with request/response and their parameters (service identifier, sub functions and data parameters). The length and data type are relevant for all data parameters; constant parameters also require a constant value. In UDS, access to certain data packets may be restricted to certain sessions or security levels. This information is also contained in the configuration data, so that the software can assure conformance to the prescribed rules.

The second important aspect of the software configuration data is that it links the diagnostic software to the application. The parameters passed by diagnostic services can be linked to variables or functions of the application software. Software generators can then generate the relevant calls.

Since AUTOSAR diagnostics is limited to the UDS and OBDII protocols, the layout of diagnostic services of these protocols is implicitly assumed and is not explicitly described in the ECUC data.

The AUTOSAR ECUC data is stored in a standardized XML format, which enables its processing in code generators.

Supplying Data to Diagnostic Testers

Diagnostic data used to parameterize generic testers must contain all information relevant to the vehicle or its ECUs from the perspective of diagnostic communication. A significant difference compared to the configuration data described above is the vehicle scope. Especially in the service area, a single diagnostic tester needs to cover a large number of different vehicles, models and variants over many model years. The resulting volume of data requires efficient mechanisms to avoid redundancy and to achieve compact storage of the necessary data.

The specification character required for configuration is not really necessary for parameterizing testers; on the contrary, it may even be advantageous for a parameterization to contain multiple equivalent alternatives, because the appropriate data can then be automatically selected at runtime. When a diagnostic tester is connected to a vehicle, it is often unclear which ECU variants and software levels are installed in the vehicle under test.

In terms of content, the diagnostic tester data differs from the configuration data in that conversion information is an essential component. The compactly coded bus messages and their parts are displayed as physical values with units at the tester.

Examples of established data formats for parameterizing diagnostic testers are the cdd format from Vector and the ISO-standardized ODX format.

Example of a Tool Chain

During diagnostic development, the following tasks are performed, which are supported by the tool chain shown in **Figure 2**.

Defining, Gathering and Coordinating Requirements

IBM DOORS is widely used among automotive OEMs as a tool for capturing and managing requirements.

Creating and Coordinating the Specification

Here, CANdelaStudio can be seamlessly integrated into the requirements-driven process chain as an authoring tool for specifying ECU diagnostics, because CANdelaStudio supports the capture and import/export of requirements. Diagnostic objects (diagnostic services, data objects, DTCs) are generated at the press of a button from the requirements, which are already formally described. These objects are each linked to an original requirement. In this way, the

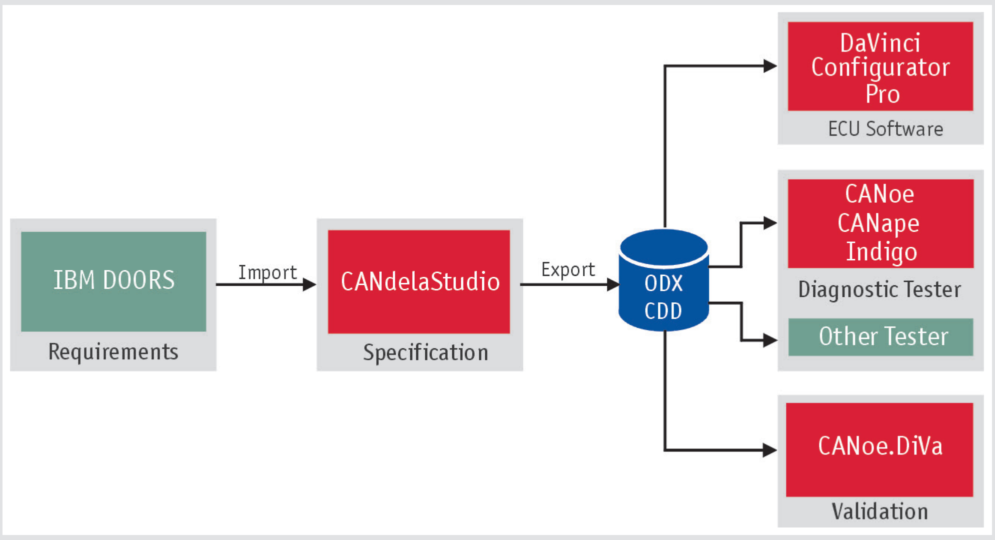


Figure 2: Tool chain of diagnostic development

user can have the imported requirements automatically adapted and synchronized to match updated requirements, and if necessary the specification can be modified. Closely interlinking requirements and specification is very advantageous in the typically iterative process, because it avoids duplicated efforts in creating and re-comparing the specification data.

The finished diagnostic specification serves as the input to subsequent steps in the process chain. CANdelaStudio saves the native diagnostic specification in cdd format, and an ODX file can also be exported at the press of a button.

Generating and Integrating ECU Software

DaVinci Configurator Pro is a tool for configuring and generating the AUTOSAR basic software and an ECU's RTE. The user imports a diagnostic specification (ODX or cdd) and generates an initial ECUC configuration from it. Afterwards, the user progressively supplements the configuration for the ECU and makes it more specific and detailed. If there is a new version of the diagnostic specification, it is easy to re-import it, and the contents are automatically merged with those of the previously created configuration. The diagnostic software for the ECU is generated based on the resulting configuration.

Testing ECU Diagnostic Software

CANoe.DiVa is used to test the diagnostic implementation in the ECU at both the supplier and the OEM. CANoe.DiVa generates an extensive set of ECU-specific test cases based on the ECU descriptions in ODX or cdd format, which are then automatically executed in CANoe. Test results are shown in detail, and the user can comment on any test cases, or group, sort and filter them according to various criteria.

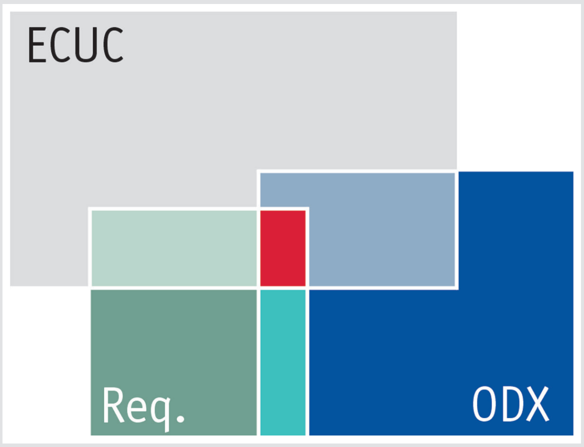


Figure 3:
Contentual similarities of the several description models

Using ECU Diagnostics

CANoe, CANape or Indigo is used as the diagnostic tester, depending on the application area. Having the CANdelaStudio specification as a common source for tester parameterization and ECU configuration ensures that the tester and ECU software match one another.

Summary

The AUTOSAR and ODX standards that have appeared in the diagnostics area in recent years complement one another well and continue to be effective in meeting objectives. Although they cover related contents, they have very different areas of focus and overlap just slightly **(Figure 3)**. The operation area of the one standard cannot be covered by the other. The AUTOSAR method is also compatible with ODX.
In practice, however, there is still the challenge of assuring consistency of the data described in the different standards over a distributed and usually iterative development process. This challenge can be overcome by a well-defined process, targeted data transfer and support by tools available on the market today.

>> Your Contact:

Germany and all countries, not named below
Vector Informatik GmbH, Stuttgart, Germany, www.vector.com

Austria and Eastern Europe
Vector Austria GmbH, Vienna, Austria, www.vector-austria.com

France, Belgium, Luxembourg
Vector France S.A.S., Malakoff, France, www.vector-france.com

Sweden, Denmark, Norway, Finland, Iceland
VecScan AB, Göteborg, Sweden, www.vector-scandinavia.com

United Kingdom & Ireland
Vector GB Ltd., Birmingham, United Kingdom, www.vector-gb.co.uk

USA, Canada, Mexico
Vector CANtech, Inc., Michigan, USA, www.vector-cantech.com

Japan
Vector Japan Co., Ltd., Tokyo, Japan, www.vector-japan.co.jp

Korea
Vector Korea IT Inc., Seoul, Republic of Korea, www.vector.kr

India
Vector Informatik India Pvt. Ltd., Pune, India, www.vector.in

China
Vector Automotive Technology (Shanhghai) Co. Ltd., Shanghai, China, www.vector-china.com

E-Mail Contact
info@vector.com

Dr. Klaus Beiter
leads a development team for the Automotive Diagnostics product line at the company Vector Informatik GmbH in Stuttgart. He is a member of the ASAM/ISO ODX working group.

Christoph Rätz
is the Director of the Diagnostics product line at the company Vector Informatik GmbH in Stuttgart.