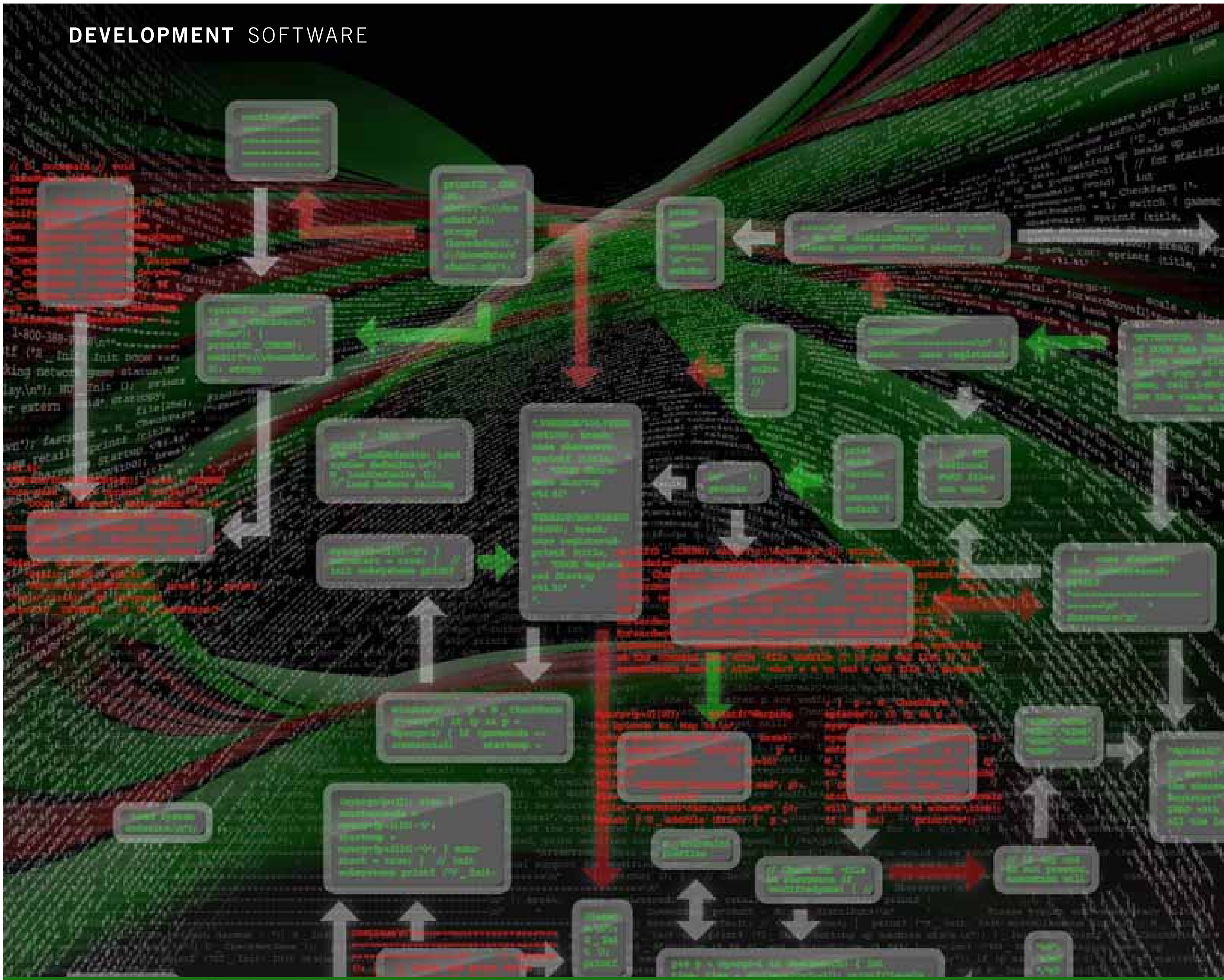


Figure © kreateur.de / Andreas Speck

AUTOSAR ARCHITECTURE AUTOMATIC CHECKING OF IMPLEMENTATION CONFORMANCE

The Autosar-based development methodology is increasingly applied during software development for the volume production. With this paradigm shift, the system‘s architecture is defined by an Autosar model that is used for generating stubs of the software components as source code and middleware providing APIs for the communication between the components. Yet, this approach does not guarantee that the manually written code – or the code generated by further tools – indeed complies with the architecture specified by the model. In a cooperation of BMW Group and CQSE GmbH it was therefore analysed, how deviations between source code and the architecture specification of the Autosar model can be identified.

AUTHORS

DR. MARTIN FEILKAS
is Founder and CEO of CQSE GmbH
in Garching near Munich (Germany).

DR. CHRISTIAN PFALLER
is Consultant for Continuous Quality
Control and Quality Improvement
Processes at CQSE GmbH
in Garching near Munich (Germany).

DR. CHRISTIAN SALZMANN
is Head of the Department for
Software Engineering in Body
Electronics and Driving Assistance
at BMW Group in Munich (Germany).

MIKE PAGEL
is Software Architect for Body
Electronics at BMW Group
in Munich (Germany).

MOTIVATION

The fast erosion of architectures in the conventional code centered development is often mentioned in the literature whereas in the analysed Autosar-based systems hardly any violation of the structural requirements could be detected. Hence, the interesting result is that some specified dependencies in the model were not used in the actual code. Usually this is the case when new interfaces were introduced but the deprecated interfaces remain in the model. Unused dependencies lead to models which are more complex than necessary but may cause the introduction of errors as well. Therefore BMW applies automatic analysis techniques to ensure the complete conformance to the architecture continuously during development. These analyses are described in the following in more detail.

CONVENTIONAL ANALYSIS OF ARCHITECTURE CONFORMANCE

Large software systems are too complex to be understood only on the source code level. This is the reason why during software development an architecture design is – implicitly or explicitly – applied, where the system is partitioned into modules or components. Since each component has to deal only with a part of the overall functionality, the system is divided into manageable units. During this process it is defined which components will communicate with each other and which way of communication should take place. Especially, it is specified among which components no communication is allowed, e.g., to ensure easier replacement of components.

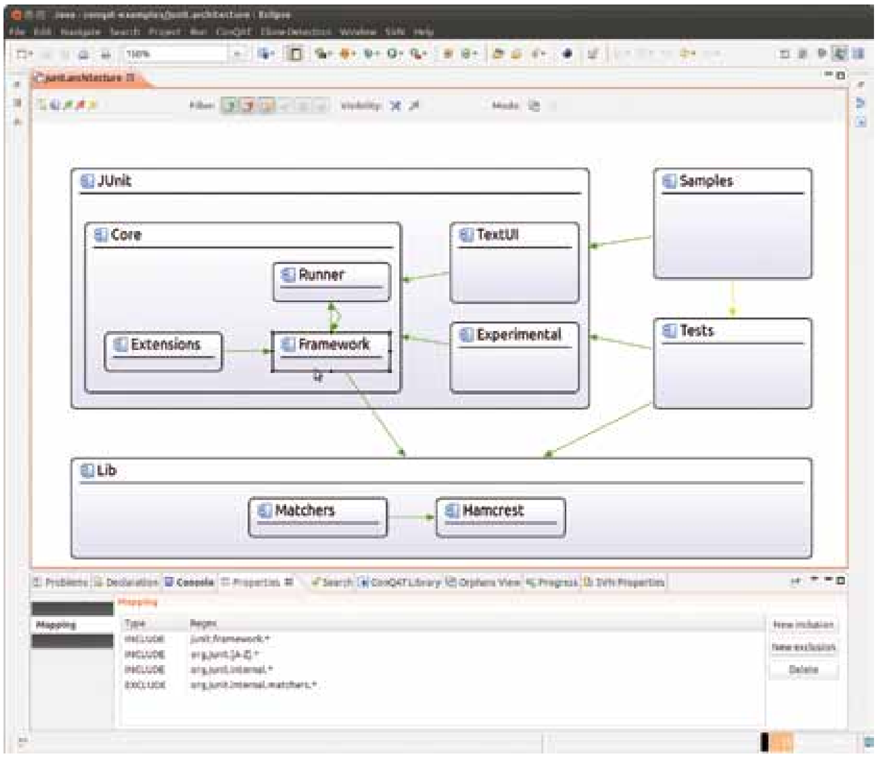
In practice, such specifications are usually created during the initial development, often in a design step before the system is implemented. During implementation, these structures are mapped to the constructs of a programming language. Studies show that within a few years of development and maintenance a heavy erosion of the intended architecture can be observed, leading to the fact that 10% to 20% of implemented dependencies do not conform to the original architecture specification [1]. On the one hand, dependencies are introduced between components which were not intended to be coupled, on the other hand, dependencies which were specified do not exist in the code anymore. Often the lack of conformance between the intended and implemented architecture results from a lack of the developer's knowledge on the architectural guidelines. But in many cases the architecture conception is enhanced mentally -by the developer team while the architecture specification is not updated. Both observations hamper the maintainability of the system, since understandability of the system is affected.

Architecture conformance analyses are applied to countervail the architecture erosion of software. These analyses check automatically if an implementation complies to the specified architecture [2, 3]. For this, the intended target architecture is specified as a model in a computer readable form. The dependencies which are implemented in the code (implemented architecture) are automatically extracted from the code and their conformance with the specified architecture is checked.

The model of the architecture specification consists of (potentially hierarchic) components. An example is given in ❶. For each component the mapping of source code files to the component is specified, ① (below). The architectural constraints are stated by edges (arrows) between components. Components which are not connected with each other must not show any communication relation in the code. If an edge connects two components, it is expected that the implementation of the source component uses the implementation of the target component (e.g., calling a function, access to a global variable, etc.). Furthermore, usage of the child components of the target component is allowed. A metamodel of the architecture specification language is given in ❷, a detailed description may be found in [4].

AUTOSAR-BASED ARCHITECTURE CONFORMANCE ANALYSIS

Automotive software development is more and more done according to the Autosar standard. In an Autosar-based system the architecture is defined in terms of Autosar models. These models allow the partitioning of an overall system in a network of components which communicate over ports. These models are used to generate a middleware layer



Type	Regex
INCLUDE	junit.framework.*
INCLUDE	org.junit.[A-Z].*
INCLUDE	org.junit.internal.*
EXCLUDE	org.junit.internal.matchers.*

❶ Example model of the intended architecture of JUnit

(runtime environment, RTE) which realises (among others) the communication between the components by an API.

Even when using Autosar, the specified and implemented architecture may deviate over time. This manifests in the following possible problems:

: Missing communication relations in the code: Components may not use the generated RTE APIs. Hence, a specified communication does not take place in the analysed code.
: Dependencies between components without using the RTE: Within C source code native C interfaces of other components may be used, bypassing the RTE. These hidden dependencies make further development more complex and hamper an individual deployment of these components.

To avoid these negative effects, an architecture analysis of the Autosar system can be performed. ❸ outlines the process auf such an analysis. First, the architecture specification is extracted from the XML description of the Autosar model and a structural view of the target architecture is generated. Second, a dependency graph is calculated by analysing the source code. For both steps the analysis tool ConQAT is used, which finally is also used to execute the conformance analysis.

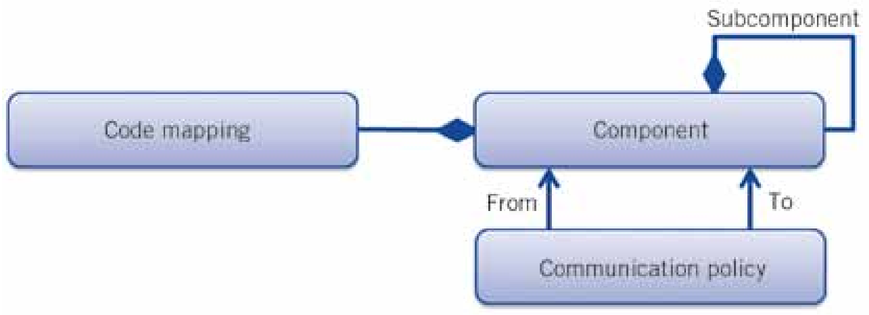
EXTRACTION OF THE STRUCTURAL ARCHITECTURE SPECIFICATION FROM THE AUTOSAR MODELS

The architecture specification used in the analysis is based on the XML files of the Autosar model. The components of the extracted architecture specification correspond directly to the Autosar components. The mapping of source code to a component may be specified in the Autosar model or it is generated according to the RTE generator used and project-specific conventions, e. g., according to the directory structure of the source code.

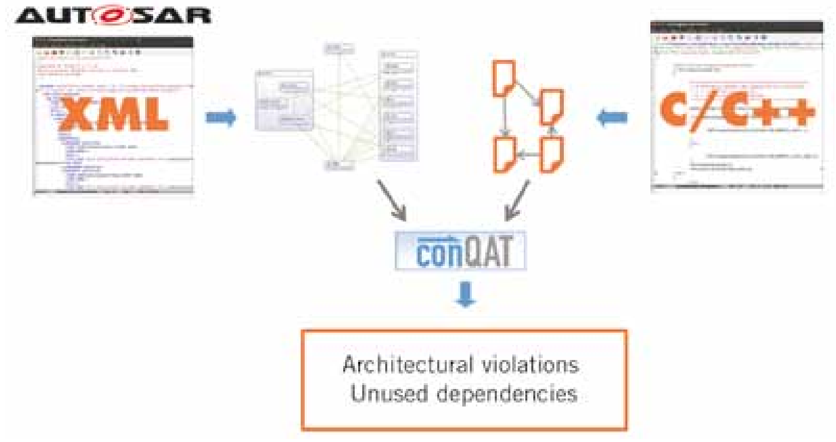
EXTRACTION OF THE IMPLEMENTED ARCHITECTURE FROM SOURCE CODE

The implemented architecture is a dependency graph where the nodes correspond to single source code files of the system. Edges describe dependencies, which exist in the code. This approach is independent from the used programming language – only the types of dependencies differ. During the analysis of C/C++ code, three dependency types are analysed:

: Include dependencies: An include dependency exists between two files if one file includes the other using the pre-processing directive #include. This dependency exists at compile time, thus the file cannot be built if the included file is not present. Since in C/C++ any kind of (relative) path is allowed for #include, it is almost impossible for the build system to prohibit unwanted dependencies.
: Declaration / implementation dependencies: In C/C++ any data structure must be declared before it is used. Typically this is done in header files which are added by an #include preprocessing directive. In general, the declaration of a function (for its later usage) can be put in any place of the code. Hence, these must be extracted and considered during the analysis.
: Indirect Autosar dependencies: Besides the before mentioned direct dependencies, with Autosar also indirect dependencies exist which result from the RTE communication. The Autosar RTE specification [5] requires that the communication must be realised by



❷ Meta-Modell of the specification of the intended architecture functions (or macros) named according to the communication ports (like Rte_Read_, Rte_Call_). Since these functions are provided by the code generator it is ensured that the naming conventions are met. With this information and information on the port connections known from the Autosar model, pairs of corresponding read/write calls are identified.



❸ Outline of the architecture analysis process based on Autosar and C/C++ code

RESULTS

The analysis identifies dependencies between fragments of code that are not allowed according to the intended architecture as stated in the Autosar model and the derived structural architecture specification. It was observed that such architectural violations rarely occur. If a violation was observed, it was mainly on dependencies to libraries, which are not covered by the Autosar model.

Beside unintended dependencies, the analysis detects dependencies which are specified between two components in the model but which do not exist in the actual code. Such unused dependencies can occur if some functionality was not implemented yet, the functionality is not covered by the analysed variant of the system, or if the planed architecture is not up-to-date anymore. Unused dependencies may also indicate errors during programming if, for example, wrong functions are called, resulting in missing dependencies.

In particular, read accesses to ports which are not written by the opposed component are regarded as critical. In such situations, the reading component expects that certain data is provided but actually only the initial value is read. Less critical but sloppy are write accesses to ports which are never read.

A regular execution of the architecture conformance analysis ensures that the intended architecture, which is specified in the Autosar model, is indeed implemented in the source code. This allows identifying possible problems early before testing and hence a more efficient removal of these problems. Furthermore, the value of the Autosar model as a map of the system is preserved. Using such an up-to-date model, especially new members in a development team may gain an overview of the system more quickly. Furthermore, impact analyses and architectural discussions can be achieved on a more solid basis. At the same time, the effort for the analysis is very low since it is executed fully automatic. If embedded in a nightly updated quality control dashboard, deviations and their fixes may be followed day by day. Such an architecture analysis may also be part of the quality examination of code supplied by third parties.

REFERENCES

[1] Feilkas, M.; Ratiu, D.; Juergens, E.: The Loss of Architectural Knowledge during System Evolution: An Industrial Case Study. Proceedings of the 17th IEEE International Conference on Program Comprehension (ICPC 09), 2009
[2] Murphy, G.; Notkin, D.; Sullivan, K.: Software reflexion models: Bridging the gap between source and high-level models. Proceedings of the Third ACM Sigsoft Symposium on Foundations of Software Engineering (FSE 95), 1995
[3] Koschke, R.; Simon, D.: Hierarchical reflexion models. Proceedings off the 10th Working Conference on Reverse Engineering (WCRE 03), 2003
[4] Deissenboeck, F.; Heinemann, L.; Hummel, B.; Juergens, E.: Flexible architecture conformance assessment with ConQAT. In Proceedings of
[5] the 32nd International Conference on Software
[6] Engineering (ICSE 10), 2010
[7] Autosar Administration: Specification of RTE, version 3.2, 2011

Offprint from ATZelektronik 03/2013,
Springer Vieweg,
Springer Fachmedien Wiesbaden GmbH