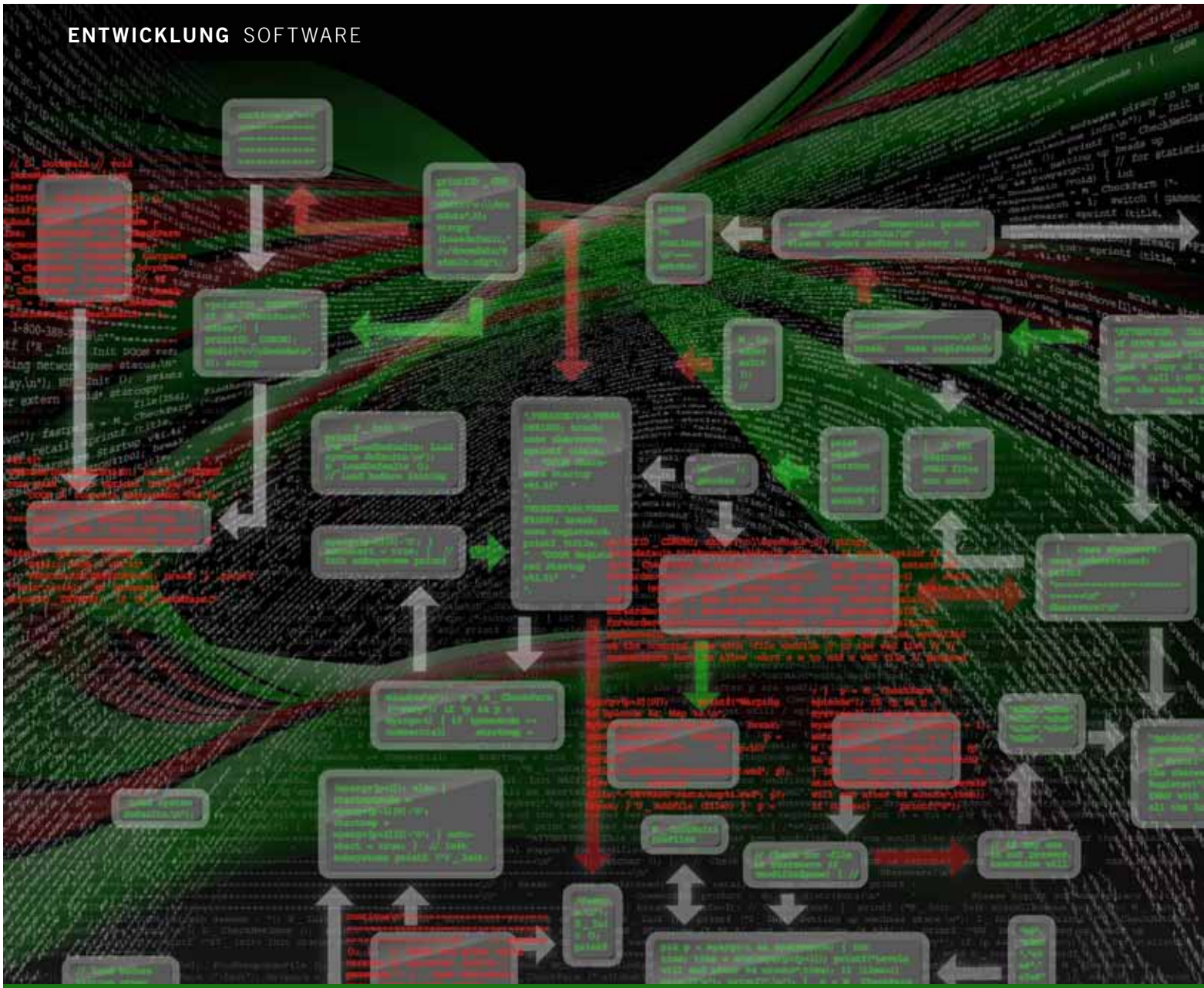


Bild © kreateur.de / Andreas Speck

AUTOSAR-ARCHITEKTUR
KONFORMITÄT DER IMPLEMENTIERUNG AUTOMATISCH PRÜFEN

Der Einsatz Autosar-basierter Entwicklungsmethodik hält immer stärker Einzug in die Softwareentwicklung für die Serie. Dieser Paradigmenwechsel, in dem die Architektur des Systems durch ein Autosar-Modell vorgegeben und über generierte Komponentenhüllen eine Middleware in Form von Code realisiert wird, lässt jedoch die Frage unbeantwortet, inwiefern sich der manuell implementierte oder durch andere Werkzeuge generierte Code an die Architekturvorgaben des Modells hält. Aus diesem Grund wurde in einer Zusammenarbeit zwischen der BMW Group und der CQSE GmbH gezielt analysiert, inwiefern sich Abweichungen zwischen den Autosar-Architekturvorgaben und dem Code identifizieren lassen.

AUTOREN

DR. MARTIN FEILKAS
ist geschäftsführender Gesellschafter der CQSE GmbH in Garching bei München.

DR. CHRISTIAN PFALLER
ist Berater für kontinuierliches Qualitäts-Controlling und Qualitätsverbesserungsprozesse bei der CQSE GmbH in Garching bei München.

DR. CHRISTIAN SALZMANN
leitet die Abteilung Software Entwicklung Karosserie und Fahrerassistenz bei BMW in München.

MIKE PAGEL
ist Softwarearchitekt im Bereich Karosserie-Elektronik bei der BMW AG in München.

MOTIVATION

In der Literatur wird auf Basis herkömmlicher Code-zentrierter Entwicklung eine starke Erosion der Architekturen beschrieben. In den analysierten Autosar-Systemen wurden auf Code-Ebene kaum Verstöße gegen die strukturellen Vorgaben gefunden. Besonders wertvoll ist jedoch, dass teilweise die im Modell spezifizierten Abhängigkeiten im Code nicht genutzt wurden. Dies liegt im Allgemeinen darin begründet, dass neue Schnittstellen geschaffen wurden und alte ungenutzt im Modell verbleiben. Da dies jedoch einerseits zu Modellen führt, die komplexer sind, als unbedingt notwendig, und andererseits auch eine potenzielle Fehlerquelle darstellt, setzt die BMW Group analytische Techniken ein, um die vollkommene Architekturkonformität entwicklungsbegleitend sicherzustellen. Diese Techniken werden im Folgenden näher beschrieben.

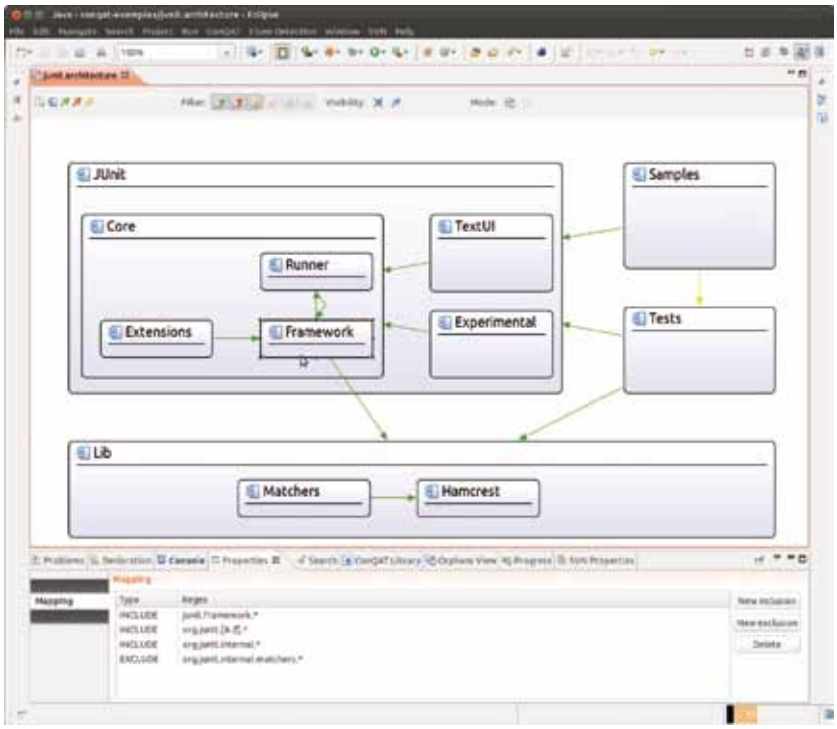
KLASSISCHE ARCHITEKTUR-KONFORMITÄTSANALYSE

Große Softwaresysteme sind zu komplex, um diese auf der reinen Code-Ebene verstehen zu können. Aus diesem Grund wird beim Entwurf von Systemen implizit oder explizit eine Architektur konzipiert, um das System in einzelne Module oder Komponenten zu zerlegen, denen eine Verantwortlichkeit für eine Teilfunktionalität übertragen wird, um so das Gesamtsystem in handhabbarere Teile zu untergliedern. Dabei wird im Rahmen der Zerlegung des Gesamtsystems festgelegt, welche Komponenten auf welche Art miteinander kommunizieren sollen – beziehungsweise welchen Komponenten explizit untersagt ist, zu interagieren, um beispielsweise eine Entkopplung zu erreichen und eine bessere Austauschbarkeit zu gewährleisten.

In der Praxis werden derartige Modelle häufig während der Initialentwicklung erstellt, oftmals in einer der Implementierung vorgelagerten Entwurfsphase. Während der Implementierung werden diese Strukturen mit den Konstrukten einer Programmiersprache nachgebildet. Studien zeigen jedoch, dass bereits innerhalb weniger Entwicklungsjahre eine starke Architekturerosion zu beobachten ist, die dazu führt, dass 10 bis 20% der im System implementierten Abhängigkeiten nicht mehr konform zur ursprünglich definierten Architektur sind [1]. Einerseits entstehen Abhängigkeiten zwischen ursprünglich entkoppelten Komponenten im Code und andererseits bestehen spezifizierte Beziehungen zwischen Komponenten auf der Code-Ebene nicht mehr. Oftmals ist der daraus resultierende Mangel an Konformität zwischen der intendierten und der implementierten Architektur auf einen Mangel an Wissen der Entwickler über die architektonischen Vorgaben zurückzuführen. In vielen Fällen beruht diese auch auf einer Weiterentwicklung der Architekturvorstellung des Entwicklerteams, die jedoch nicht in die Architekturspezifikation übernommen wurde. Beide Phänomene sind als Problem für die Wartbarkeit und Weiterentwickelbarkeit des Systems zu betrachten, da sie die Verständlichkeit des Systems beeinträchtigen.

Um diesem Verfall der Softwarearchitektur entgegenzuwirken, werden Architekturkonformitätsanalysen eingesetzt. Diese prüfen automatisch, ob eine Implementierung die Vorgaben einer spezifizierten Architektur erfüllt [2, 3]. Hierfür werden die Architekturvorgaben in Form eines Modells der Soll-Architektur in maschinenlesbarer Form spezifiziert. Die im Code implementierten Abhängigkeiten (Ist-Architektur) werden schließlich automatisch extrahiert und auf Konformität mit der Soll-Architektur überprüft.

Das Modell der Soll-Architektur besteht aus (potenziell hierarchischen) Komponenten. Ein Beispielmodell ist in ❶ zu sehen. Für die Komponenten wird jeweils ein Code-Mapping, ① (unten), spezifiziert, das angibt, welche Code-Dateien die Implementierung einer Komponente umfassen. Die Architekturvorgaben werden durch Kanten (Pfeile) zwischen den Komponenten zum Ausdruck gebracht. Es gilt der Grundsatz, dass Komponenten, die nicht verbunden sind, keine Kommunikationsbeziehung im Code aufweisen dürfen. Verbindet eine Kante zwei Komponenten, so wird erwartet, dass die Implementierung der Quellkomponente die Implementierung der Zielkomponente nutzt (zum Beispiel durch Aufruf einer Funktion, Zugriff auf eine globale Variable etc.). Auch der Zugriff auf die Unterkomponenten der Zielkomponente ist über diese Kante erlaubt. Ein Meta-Modell ist in ❷ dargestellt, eine genaue Beschreibung des Soll-Architekturmodells ist in [4] zu finden.



❶ Beispielmodell einer Sollarchitektur für JUnit

ARCHITEKTURKONFORMITÄTS-ANALYSE AUF BASIS VON AUTOSAR

Im Bereich der Softwareentwicklung für Automobile ist die Softwareentwicklung zunehmend durch den Autosar-Standard geprägt. In Autosar-basierten Systemen ist die Architektur der Systeme in Form von Autosar-Modellen definiert. Diese Modelle erlauben die Zerlegung eines Gesamtsystems in ein Netzwerk von Komponenten, die über Ports miteinander kommunizieren. Diese Modelle werden dazu verwendet, eine Middleware-Schicht (RTE: Runtime Environment) zu generieren, welche unter anderem die Kommunikationsbeziehungen zwischen den Komponenten in Form einer API realisiert.

Auch bei Nutzung von Autosar kann es jedoch zu einem Auseinanderdriften der Soll- und der Ist-Architektur kommen. Dies äußert sich in folgenden möglichen Problemen:

: Fehlende Kommunikationsbeziehungen auf Code-Ebene: Komponenten nutzen die in die RTE-generierten APIs nicht. Somit findet die im Modell spezifizierte Kommunikation im analysierten Code nicht statt.

: Abhängigkeiten zwischen Komponenten ohne Nutzung der RTE: Auf Ebene des C-Codes ist es möglich, die nativen C-Schnittstellen anderer Komponenten zu nutzen und damit die RTE zu umgehen. Durch diese verdeckten Abhängigkeiten wird die Weiterentwicklung erschwert und ein getrenntes Deployment unmöglich.

Um diesen Auswirkungen zu begegnen, kann eine Architekturanalyse auch auf Autosar-Systemen eingesetzt werden. ❸ skizziert den Ablauf einer Architekturanalyse auf Basis von Autosar. Hierbei werden einerseits die Architekturvorgaben aus der XML-Beschreibung des Autosar-Modells extrahiert und eine rein strukturelle Sicht auf die Soll-Architektur generiert. Darüber hinaus wird ein Abhängigkeitsgraph aus dem Quellcode berechnet. Für beide Schritte wird das Werkzeug ConQAT eingesetzt, das schließlich auch die Konformitätsanalyse durchführt.

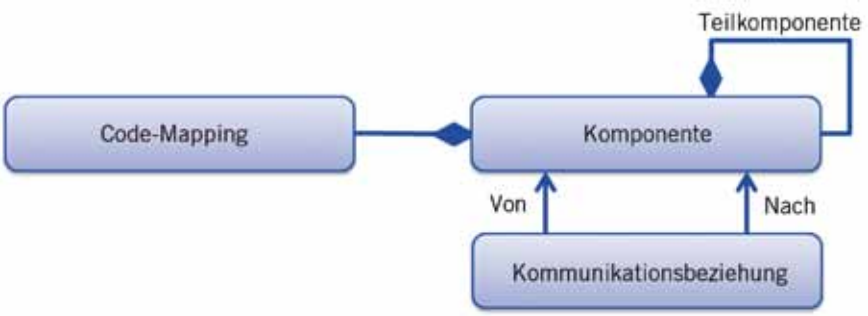
EXTRAKTION DER STRUKTURELLEN ARCHITEKTUR AUS AUTOSAR-MODELLEN

Grundlage für die zur Analyse verwendete Spezifikation der Soll-Architektur ist das in XML-Dateien vorliegende Autosar-Modell. Die Komponenten der Soll-Architektur entsprechen den Autosar-Komponenten. Die Zuordnung von Programmcode zu einer Komponente (Code-Mapping) ist entweder in einer Autosar-Komponente spezifiziert oder erfolgt abhängig vom eingesetzten RTE-Generator und projektspezifischen Konventionen, etwa anhand der Verzeichnisstruktur.

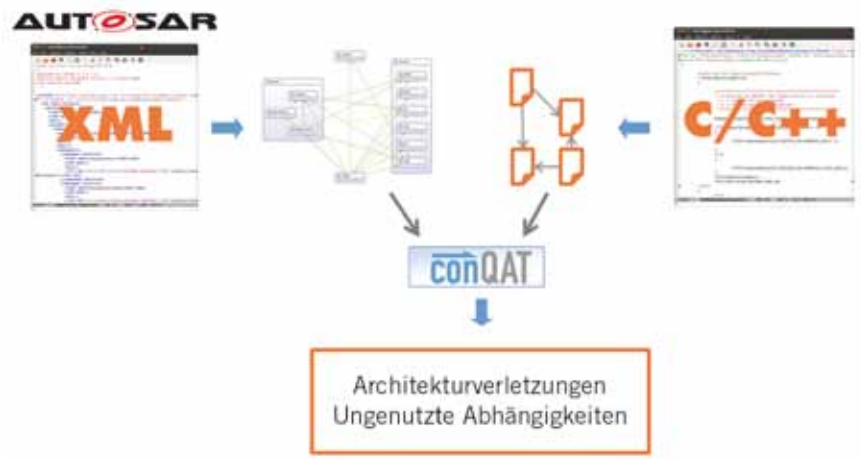
EXTRAKTION DER IST-ARCHITEKTUR AUS DEM QUELLCODE

Die Ist-Architektur besteht aus einem Abhängigkeitsgraphen bei dem die Knoten durch die einzelnen Quelldateien des Systems gebildet werden. Die Kanten beschreiben die bestehenden Abhängigkeiten im Code. Grundsätzlich ist der Ansatz unabhängig von der zugrundeliegenden Programmiersprache, lediglich die Arten der Abhängigkeiten unterscheiden sich. Für die Analyse von C/C++-Quellcode werden drei Arten von Abhängigkeiten berücksichtigt:

: Include-Beziehungen: Eine Include-Beziehung besteht zwischen zwei Dateien, wenn eine die andere mittels der Präprozessordirektive „#include“ einbindet. Dies ist eine Abhängigkeit,



❷ Meta-Modell einer Sollarchitekturbeschreibung



❸ Schematischer Ablauf einer Architekturanalyse auf Basis von Autosar und C/C++ Code

die zur Compile-Zeit besteht, das heißt, eine Datei kann nicht übersetzt werden, wenn die eingebundenen Dateien nicht vorhanden sind. Da in C/C++ beliebige (relative) Pfade für include erlaubt sind, ist es durch das Build-System fast nicht möglich, ungewollte Abhängigkeiten komplett auszuschließen.

: Deklarations-/Implementierungsbeziehungen: In C/C++ wird jede benutze Datenstruktur und aufgerufene Funktion in den Quelldateien zuvor deklariert. Typischerweise erfolgt dies in Header-Dateien, die durch den Präprozessor per „#include“ eingefügt werden. Grundsätzlich ist die Deklaration (und spätere Verwendung) einer Funktion jedoch an jeder Stelle im Code möglich, somit sind diese explizit zu extrahieren und in der Analyse zu berücksichtigen.

: Indirekte Abhängigkeiten aus Autosar: Neben den bisher betrachteten direkten Abhängigkeiten gibt es in Autosar auch indirekte Abhängigkeiten, die durch die Kommunikation über die RTE entstehen. Die Autosar RTE-Spezifikation [5] schreibt vor, dass die Kommunikation durch Funktionen (oder Makros) realisiert sein muss, deren Namen sich aus den verwendeten Kommunikationsports ableiten (wie Rte_Read_, Rte_Call_). Da diese Funktionen durch den Generator bereitgestellt werden, ist sichergestellt, dass diese Namenskonvention eingehalten wird. Dadurch und durch die aus dem Autosar-Modell bekannten Port-Verbindungen lassen sich passende Paare von lesenden und schreibenden Aufrufen erkennen.

ERGEBNISSE

Die Analyse identifiziert Abhängigkeiten zwischen Code-Fragmenten, die anhand des Autosar-Modells und der abgeleiteten Soll-Architektur nicht erlaubt sind. Es wurde festgestellt, dass diese Art von Architekturverletzungen in Autosar-Systemen kaum vorkommt, wenn überhaupt, dann in Form von Abhängigkeiten zu Basissoftware oder Bibliotheken, die nicht vom Autosar-Modell abgedeckt sind.

Neben den nicht im Modell vorgesehenen Abhängigkeiten findet die Analyse auch Abhängigkeiten, die im Modell zwischen zwei Komponenten bestehen, jedoch nicht im Code. Solche unbenutzten Abhängigkeiten können bedeuten, dass Teile des geplanten Funktionsumfangs noch nicht oder nicht in der vorliegenden Variante realisiert sind, beziehungsweise dass die geplante Architektur mit der aktuellen Situation nicht mehr zusammenpasst. Sie können aber auch Hinweise auf Programmierfehler geben, wenn etwa falsche Funktionen aufgerufen werden und daher keine Verbindung besteht.

Als besonders kritisch wären im Code realisierte lesende Zugriffe auf Ports, auf deren gegenüberliegender Seite jedoch keine Schreibzugriffe stattfinden. In diesem Fall verlässt sich die lesende Seite darauf, dass bestimmte Daten zur Verfügung gestellt werden, diese werden jedoch nicht wie erwartet bereitgestellt, sondern es wird lediglich immer der Initialwert gelesen. Weniger kritisch aber dennoch unschön sind schreibende Port-Zugriffe, die jedoch niemals von anderen Komponenten gelesen werden.

Die regelmäßige Durchführung eines Architekturabgleichs stellt sicher, dass die im Autosar-Modell vorgegebene Architektur auch tatsächlich im Code umgesetzt wird. Dadurch lassen sich zum einen mögliche Probleme schon vor der Testphase identifizieren und effizient beheben. Zudem bleibt der Wert des Autosar-Modells als „Landkarte des Systems“ erhalten, was gerade neuen Entwicklern eine schnellere Orientierung im System ermöglicht. Auch Impact-Analysen und Architekturdiskussionen können dadurch auf einem fundierten Niveau durchgeführt werden. Gleichzeitig ist der Aufwand durch die vollständige Automatisierung des Verfahrens sehr gering. Eingebettet in einen nächtlich aktualisierten Qualitätsleitstand, lassen sich Abweichungen und deren Beseitigung tagesaktuell verfolgen. Auch als Teil einer Qualitätsprüfung bei Abnahme von zugeliefertem Code kann diese Analyse dienen.

LITERATURHINWEISE

[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 the 32nd International Conference on Software Engineering (ICSE 10), 2010

[5] Autosar Administration: Specification of RTE, version 3.2, 2011

Sonderdruck aus ATZelektronik 03/2013,
Springer Vieweg,
Springer Fachmedien Wiesbaden GmbH