ENTWICKLUNG EINER RTE FÜR DIE ASIL-EINSTUFUNG

Wie wird das AUTOSAR-RTE ASIL-D-konform?

In vielen aktuellen Entwicklungsprojekten wird die AUTOSAR-RTE (Runtime Environment) noch manuell kodiert, da AUTOSAR-Tool-Hersteller den RTE-Generator nicht für ASIL X zertifiziert haben. Dabei lässt die ISO 26262 eine Reihe von Maßnahmen zu, mit denen in sicherheitskritischen Projekten eine generierte RTE auch ohne eine Qualifizierung durch den Tool-Hersteller verwendet werden kann. Die denkbar schlechteste Lösung ist dabei die Kodierung der RTE von Hand.

Bei der Software-Entwicklung können Code-Generatoren, wie Compiler oder der RTE Generator, fehlerhaft sein und deshalb zu Fehlern im Produkt führen. Daher müssen Maßnahmen ergriffen werden, die verhindern, dass Fehler in das Produkt gelangen, die sich gefährlich auswirken können. Dieses Ziel lässt sich auch durch die Verifikation der erzeugten Software erreichen.

Am Anfang jeder Sicherheitsbetrachtung steht die Gefahrenanalyse. Dabei werden Situationen aus der Sicht des Fahrzeuges bzw. des Fahrers untersucht, in denen vom betrachteten System eine Gefahr für Mensch und Umwelt ausgeht. Die Gefahren werden im Hinblick auf Häufigkeit der Situation, Beherrschbarkeit durch beteiligte Personen und Schadenshöhe bewertet und klassifiziert. In der ISO 26262 ist diese Einstufung geregelt und wird als Automotive Safety Integrity Level – kurz ASIL – bezeichnet. Es gibt dabei vier Stufen von „A“ bis „D“, wobei die höchste Einstufung „D“ im Falle eines gefährlichen Ausfalls lebensbedrohliche Verletzungen zur Folge haben kann. Basierend auf dieser Einstufung beschreibt die Norm Maßnahmen, die sicherstellen sollen, dass das vom System ausgehende Risiko in einem vertretbaren Rahmen bleibt.

Nachdem bei der Gefahrenanalyse das gesamte System betrachtet wird, muss bei der Erstellung der Systemarchitektur die ASIL-Einstufung auf die einzelnen Elemente des Systems heruntergebrochen werden, unter anderem auch auf die Komponenten der Software. Dabei sollte darauf geachtet werden, dass möglichst wenige Komponenten im kritischen Pfad zu finden sind. Als kritischen Pfad bezeichnet man denjenigen Teil der Daten- und Kontrollflüsse, deren Ausfall zu einem der oben analysierten gefährlichen Ausfälle führen kann.

AUTOSAR und die RTE

Die Algorithmen, die die eigentliche Funktion des Steuergerätes ausmachen, werden dabei in der Regel als AUTOSAR-Softwarekomponenten (SWC) realisiert. Die benötigten Schnittstellen zu anderen SWCs, AUTOSAR-Basissoftware (BSW) und Treibern stellt das Runtime Environment (RTE) zur Verfügung. Diese Kommunikationsschicht wird dabei komplett aus einer XML-basierten Konfiguration (arxml) generiert. In der Konfiguration sind unter anderem die Kommunikationsendpunkte, Kommunikationsarten und auch die Nachrichtentypen beschrieben. Neben der Kommunikation bestimmt die RTE außerdem wesentlich das zeitliche Verhalten der SWCs.

Hat nun eine SWC eine ASIL-Einstufung erhalten, so müssen alle Teile der RTE, die die Funktionsweise dieser SWC beeinträchtigen können, auch entsprechend der ASIL-Einstufung entwickelt werden. Gleichzeitig müssen die verwendeten Werkzeuge, wie zum Beispiel der RTE-Generator, entsprechend qualifiziert sein. Es gibt nach derzeitigem Kenntnisstand noch keinen RTE-Generator mit einer Zertifizierung analog zu ISO 26262.

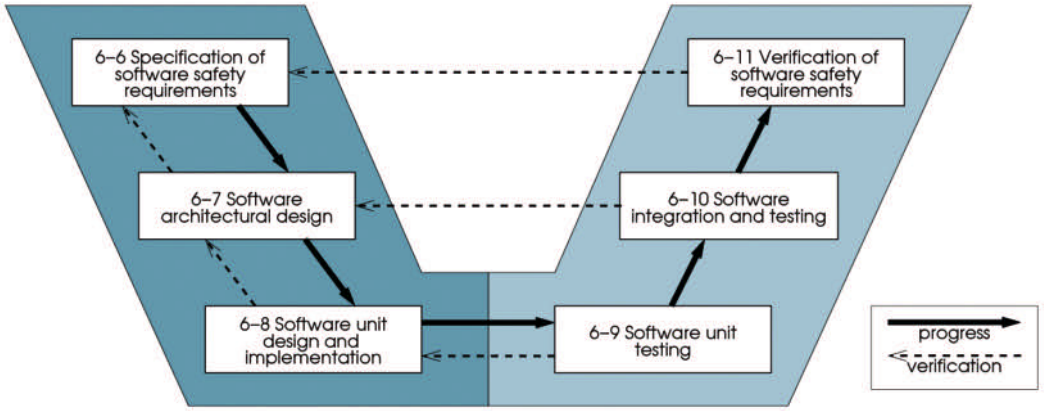

Bild 1: Vorgehensmodel zur Software-Entwicklung nach ISO 26262.

Eine Möglichkeit ist nun, die RTE mit den für die ASIL-Einstufung notwendigen Maßnahmen bei der Anforderungsanalyse, Design, Implementierung und Verifikation von Hand zu entwickeln; oder man qualifiziert den RTE-Generator selbst. Als dritte Alternative bietet es sich an, die nötigen Maßnahmen für die manuelle Implementierung auf den generierten Code zu übertragen.

Der dritte Weg

Betrachtet man das in der Norm beschriebene Vorgehensmodel (**Bild 1**), so wird beim dritten Ansatz der Schritt „6-8 Software unit design and implementation" durch das Implementieren der Software Unit mithilfe des Generators ersetzt. Dem generierten C-Code vertraut man am Ende genauso wenig wie einem von Menschen erstellten C-Programm.

Im ersten Schritt muss daher sichergestellt werden, dass der generierte Code dem Software Unit Design gerecht wird. Dies kann man erreichen, indem man beispielsweise mit Reviews überprüft, ob die Konfiguration das Design widerspiegelt. Anschließend muss durch Code-Reviews nachgewiesen werden, dass der erzeugte Code der AUTOSAR-Konfiguration entspricht.

Für den Schritt 6-8 sieht die Norm eine Reihe von Maßnahmen vor, die man nicht außer Acht lassen sollte. Hier ist zu berücksichtigen, dass die Maßnahmen dafür gedacht sind, solche Fehler zu vermeiden, die üblicherweise von menschlichen Programmierern gemacht werden. Für RTE-Generatoren erscheinen daher andere Maßnahmen sinnvoller. **Tabelle 1** listet die in der Norm genannten Methoden auf.

Die Einhaltung der Methoden kann mit statischen Testwerkzeugen, wie beispielsweise PClint oder QA-C, auch im generierten Code überprüft werden. Generierte RTE-Konstrukte, die die Regeln verletzen, sind in der Konfiguration entsprechend zu meiden oder müssen durch ein Code-Review abgesichert werden.

Einige Maßnahmen sind darin begründet, dass sie zu unleserlichem Code führen und als Folge häufiger zu Programmierfehlern. Ein Generator macht diese Fehler nicht, er verliert nicht den „Überblick" im generierten Code, sondern höchstens in seinen internen Datenstrukturen - und dieses unabhängig davon, ob die genannten Maßnahmen eingehalten werden oder nicht. Es sei auch angemerkt, dass der vom C-Compiler erzeugte Maschinencode voll von „unconditional jumps" ist, aber kein Mensch würde hier auf die Idee kommen, diese zu verbieten. Für die RTE gibt es noch keine Empfehlung für mögliche Einschränkungen; dieses liegt daher im Ermessen des Projektes.

Die Maßnahmen zur Verifikation im rechten, aufsteigenden Ast des V-Modells aus Tabelle 1 werden in vergleichbarer

	Methods	ASIL			
		A	B	C	D
1a	One entry and one exit point in subprograms and functions	++	++	++	++
1b	No dynamic objects or variables, or else online test during their creation	+	++	++	++
1c	Initialization of variables	++	++	++	++
1d	No multiple use of variable names	+	++	++	++
1e	Avoid global variables or else justify their usage	+	+	++	++
1f	Limited use of pointers	o	+	+	++
1g	No implicit type conversions	+	++	++	++
1h	No hidden data flow or control flow	+	++	++	++
1i	No unconditional jumps	++	++	++	++
1j	No recursions	+	+	++	++

Tabelle 1: Empfehlungen zur Einschränkung des Sprachumfangs nach ISO 26262.

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test	+	+	+	++
1d	Resource usage test	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable	+	+	++	++

Tabelle 2: Empfehlung zu Unit-Testmethoden nach ISO 26262.

Weise auf den generierten Code angewandt. Die Tests sollen sicherstellen, dass Fehler in der Implementierung der erzeugten RTE, wie etwa Pufferüberläufe, erkannt werden. Für „6-9 Software unit testing" empfiehlt die Norm die in der Tabelle 2 dargestellten Maßnahmen zur Verifikation der Software-Einheiten. Alle Maßnahmen können auch auf eine generierte RTE angewendet werden.

Bei diesen Maßnahmen sollte jedoch beachtet werden, dass Generatoren sich im Vergleich zum Menschen deterministischer verhalten. Es ist daher möglich, für unterschiedliche RTE-Konstrukte entsprechende Klassen von Testfällen zu bilden und nicht das gleiche RTE-Konstrukt immer wieder zu überprüfen, nachdem es einmal in ausreichendem Maße verifiziert wurde. Die Ressourcen sollten eher dazu verwendet werden, jede dieser Klassen intensiver zu testen.

Bei der RTE hat man mit der XML-Konfiguration und der zugrundeliegenden AUTOSAR-Spezifikation eine formale Spezifikation der RTE und kann diese auch für die Verifikation nutzen.

Drum prüfe, wer sich bindet ...

Die RTE bildet das Bindeglied zwischen den SWCs. Um diese Aufgabe zu erfüllen, bietet sie unterschiedliche Dienste an. Dabei sollte die Verwendung der Dienste in der Architektur- und Designphase eingeschränkt werden, zum einen, um kritische Fehler wie etwa „Dead locks" von vornherein auszuschließen, zum anderen, um auch den Aufwand für die Verifikation zu verringern, da nur Konstrukte verifiziert werden müssen, die auch verwendet werden.

Ein Beispiel für kritische Dienste, die zu „Dead locks" führen können, sind die blockierenden Varianten der Sender-Receiver- bzw. der Client-Server-Kommunikation. Darüber hinaus entsteht bei dieser Art von Kommunikation eine zeitliche Abhängigkeit zwischen unterschiedlichen Tasks. Dies erschwert die Analyse des zeitlichen Verhaltens der Gesamtsoftware oder macht diese Überprüfung sogar unmöglich. Die Abhängigkeit kann auch dazu führen, dass für andere SWCs mit einer hohen zeitlichen Bindung eine höhere ASIL-Einstufung erforderlich wird.

Im Gegensatz zur Programmiersprache C, in der mit MISRA-C bereits ein Standardkatalog vorhanden ist, der den Sprachumfang einschränkt, gibt es einen vergleichbaren Katalog für die RTE nicht. In der Architekturphase sollte daher ein solcher Katalog erarbeitet werden: Er sollte beschreiben, welche Dienste der RTE nutzen darf und welche nicht. Die entsprechenden Konstrukte dürften anschließend in der RTE-Konfiguration nicht vorkommen; sie könnten durch Prüfen der XML-Datei ausgeschlossen werden.

Was macht den Unterschied aus?

Im Vergleich zur handkodierten RTE erspart das beschriebene Vorgehen den eigentlichen Aufwand für die Codierung. Darüber hinaus sind Testergebnisse innerhalb einer Klasse von RTE-Konstrukten übertragbar. In der Folge sinkt auch der benötigte Testaufwand bzw. kann die gewonnene Zeit dazu genutzt werden, die Testtiefe für die einzelnen Klassen zu erhöhen.

Das Vorgehen ist vergleichbar mit einer Toolqualifizierung. Teile der Testergebnisse und der eingeschränkte Umfang der RTE können auf Nachfolgeprojekte übertragen werden, zumindest solange sich die Rahmenbedingungen nicht wesentlich verändern. Es wird sozusagen eine Toolqualifizierung im Projektkontext durchgeführt.

Fazit

Es ist nicht notwendig, viel Geld in die Zertifizierung eines RTE-Generators zu stecken, nur um ihn in einem sicherheitskritischen Umfeld einzusetzen. Mit etwas mehr Aufwand bei der Verifikation der generierten Ergebnisse kann genauso gut nachgewiesen werden, dass sich keine gefährlichen Fehler nur aufgrund eines Werkzeugs in die Software einschleichen. Auch wenn das Thema Toolqualifizierung noch für viel Diskussionsstoff sorgen wird, hat man unabhängig davon jederzeit die Möglichkeit, das eigentliche Produkt – das System inklusive der Software – ausreichend zu qualifizieren. (oe)

Dr. Christian Wawersich ist Consultant bein der Method Park Software AG. Schwerpunkte seiner Arbeit sind System Engineering, AUTOSAR und Qualitätssicherung in Software-Entwicklungsprozessen. Er ist zertifizierter intacs Provisional Assessor for Automotive SPICE.
Christian.Wawersich@methodpark.de

Bernhard Sechser ist bei der Method Park Software AG als Principal Consultant SPICE & Safety und intacs Principal Assessor tätig.
Bernhard.Sechser@methodpark.de

Method Park Software
www.methodpark.de