Decompositional Safety Analysis of Automotive Software: From Functional Safety to Component Testing

Name Wojciech Mostowski And Mohammad Reza Mousavi

Abstract text Strong safety requirements for automotive software are natural and are regulated by industry standards like ISO 26262 [3]. Given ASIL levels for system functionalities, measures have to be taken to ensure the correctness of the software of the associated components, and one of the assurance methods employed is testing. However, certain types of resulting safety requirements on software components are rather difficult to be established through classical testing, for example that a software fault in one component does not propagate to another component or the top-level of the system, or similarly that there are no certain types of interactions between components, some of which are possibly faulty.

In the AUTO-CAAS project [1] we employ model-based testing as a complementary assurance method to classical testing, in particular we use the QuviQ's QuickCheck model-based testing tool and apply it to automotive software implemented in C based on the AUTOSAR standard [2]. Higher-level safety requirements are first translated and decomposed into requirements on system interactions, this part of the process is inevitably mostly manual. To verify the resulting requirements on system and component interactions, we turn to the distinguished ability of QuickCheck to mock certain parts of the system under test and trace the calls to the mocked components. Consequently, interactions within the system can be tested out automatically based on high-precision specifications. Additionally, we can inject faults into the mocked components to analyse propagation and effects of faults.

The injected faults can be based on known faults or non-conformances resulting from ambiguities in the basic software specifications. In the big picture this allows us to test interaction properties already in the pre-integration phase of system development, but already on the level of concrete implementations.