BSSE System and Software Engineering

Abstract for FASTII-CCN2 Final Presentation

December 2020

FASTII: Automated Source-code-based Testing, Improvement

FASTII-CCN2: Advanced Automated Testing Techniques

A challenging issue of static and dynamic analysis is the occurrence of false positives (FP, reports on non-existing errors). In case of abstract interpretation – a static analysis method – two major sources of FP were found: lost information of the context increasing with size and complexity of the source code and missing context information on lower system levels when partitioning the code to reduce the complexity for analysis. In case of automated source code-based testing – a method of dynamic analysis focusing on units / function level by auto-generation of (random) test data – missing information on the context of units is an issue, too.

The activities of FASTII-CCN2 focused on auto-generation of test data on system level aiming to investigate the expected reduction on the number of reports due to sufficient context information on the lower level functions. For two system interfaces test data were automatically generated: telecommands (TC) from an EDS/XML specification, and test data for the external data interfaces from C-based data and type specifications. The data were fed into the system while executing all the tasks on a host environment, using the instrumentation for error detection from unit testing.

A significant reduction of reports – including true positives (TP, reports on existing errors) and false positives – was observed from about 800 (without fault injection) and 1600 (with fault injection) to about 40 in the system-level scenario compared to source-code based unit testing. The figures on number of reports issued by 4 static analysers are in the range of 800 – 28,000. Due to the reduction of reports and indicators for reports, having a good chance to turn out as TP, a number of errors was identified at little effort – including some edge cases.

By TC injection coverage figures of 60% for blocks and 70% for conditions were achieved at 100% mode and task coverage and full coverage of supported TC. As analysis shows, the missing block and condition coverage is a matter of test data generation for the external data interface to other components in the spacecraft. To a major part the processing of acquired external data is based on state machines, of which the logic is not visible in the pure C-code used as specification. This suggests to base (random) test data generation on a higher level than C-code. By merging code coverage from unit testing 78% block coverage and 86% condition coverage are achieved.

The found TP and sources of FP in the 40 reports will be discussed.