Paper ID: 11940
oral
student

41st STUDENT CONFERENCE (E2)
Student Team Competition (3)

Author: Mr. Bastian Bätz
Institute of Space Systems, University of Stuttgart, Germany, baetz@irs.uni-stuttgart.de

Mr. Seyyed Mahdi Najmabadi
Institute of Space Systems, University of Stuttgart, Germany, sm.najmabadi@gmail.com
Mrs. Zahra Tavakoli
Institute of Space Systems, University of Stuttgart, Germany, tavakoza@studi.informatik.uni-stuttgart.de
Mr. Claas Ziemke
Private, Germany, claas.ziemke@gmx.net

MODERN SOFTWARE QUALITY CONTROL METHODS AND TOOLS APPLIED TO A UNIVERSITY SMALL SATELLITE ON-BOARD SOFTWARE PROJECT

Abstract

Because of the mission criticality of on-board software, quality control is essential in satellite on-board software development. In the course of an university small satellite product this becomes even more important due to the fact that mostly undergraduate students are working on the software development. Standard methods of quality control in software engineering include source-code revision control, unit-testing and code coverage analysis. In this paper the tools and methods used for the on-board software testing for the small satellite Flying Laptop, currently under development at the Institute for Space Systems of the University Stuttgart, are described. Key features of the on-board software are the usage of the operating system RTEMS, the high-level programming language C++, a central variable data-pool and a plug-in like infrastructure for the communication with the peripheral hardware. These plug-ins are called device-handlers and are tested and verified with the methodologies described in this paper. In order to automate the unit-testing and code coverage analysis a continuous integration infrastructure has been set up. The continuous integration tool Hudson-CI monitors the git repository used for software revision control. When Hudson-CI detects a change in the source-code repository it automatically triggers a build of the software. If the build is successful Hudson-CI then generates a set of unit-tests from an SQL database. In this database all information needed to test the software is stored. The continuous integration tool queries the databases to generate an XML report, which then is transformed directly to C++ source-code for unit-testing through an XSLT processor. When the unit-testing source-code is generated the continuous integration tool executes the unit-tests and generates an unit-testing report together with the according code coverage data. The log files of the unit-tests and the code-coverage analysis results then can be inspected through the web interface of the continuous integration tool. If any something goes wrong during this procedure, a failure report is sent to the on-board software architect by e-mail. Finally the well-tested device-handlers are ready for verification on a real-time satellite system-simulator. The infrastructure described in this paper allows a team of undergraduate students that collaborate in such a complex project to focus on the implementation rather then debugging and testing. Furthermore through the centralized SQL database information consistency is enforced and through the automatic testing and failure reporting possible sources of bugs can be spotted early and the students can be lead efficiently.