Deduced Software Design Principles from Experiences with the Technical Debt in Reused Software

Olaf Maibaum[1], Thomas Terzibaschian[2], Christian Raschke[3], Andreas Gerndt[1]

[1] German Aerospace Center (DLR), Simulation and Software Technology
Lilienthalplatz 7, 38108 Braunschweig, Germany
Phone: +49 531 295 2974, Mail: olaf.maibaum@dlr.de

[2] German Aerospace Center (DLR), Institute of Optical Sensor Systems
Rutherford-Str. 2, 12489 Berlin, Germany
Phone: +49 30 67055 586, Mail: thomas.terzibaschian@dlr.de

[3] Astro- und Feinwerktechnik Adlershof GmbH
Albert-Einstein-Straße 12, 12489 Berlin, Germany
Phone: +49 30 6392 1053, Mail: c.raschke@astrofein.com

Abstract: Software reuse conserves software design decisions for the future, but technology evolves over time. Previous design decisions thereby become technical debt in designs for future projects in the case of software reuse. This may result in risk and cost increases for future projects. The small satellite BIROS is an example of such a project and is examined in this paper. The reused software prohibits state-of-the-art test techniques without significant modification to the software architecture, which is not a software reuse. The necessary changes in the software architecture are presented in this paper and how it protects against faults that arose during the system test and commissioning phase.

1. INTRODUCTION

The AOC subsystem of the BIROS satellite [1] has a long history. The first lines of C++-code in the software were written in the year 2000 for the BIRD ACS. In 2008, the BIRD ACS was reused in the TET-1 satellite. For TET-1 the setup process of the software was restructured, the hardware related drivers on the IO level of the software was replaced, and new control modes were introduced. These modifications are caused by changes in the sensor and actuator hardware as well as lessons learned from BIRD. The BIROS AOCS software, reused from TET-1 [2], has been extended for several experiments like qualification of a thruster system, fast slew maneuvers with high torque wheels, the AVANTI experiment for the autonomous formation flight with the BEESAT-4 as remote target ejected from BIROS, and an optical downlink.

In the year 2000, the development team's knowledge and the state of test techniques led to design decisions in the software architecture which did not match state-of-the-art techniques used by software development today. Due to software reuse and an unchanged software architecture, the design decisions from the BIRD software are conserved. This complicates adding new functionality and the discovery of software faults in the space-proven application software. Testing with a continuous integration test approach is not possible. This disadvantage is a technical debt that originating from the BIRD software.

Unit tests are an essential part of the continuous integration test process. In the year 2000 such software development processes were not state-of-the-art yet. E.g., the development of the cpp-Unit framework as a unit test framework for C++ started in the year

2000. The normal approach was based on functional tests, which were done for BIRD with the controller modes. Therefore, the chosen software architecture for BIRD did not support unit tests with current unit test frameworks. One positive aspect in the software architecture is the coupling of software components by a component manager, which allows the replacement of software components. Unfortunately, the interface design did not allow any replacement of the software components by test stubs. The interfaces for the serial buses were designed to allow the replacement by stubs, but all these interfaces are fixed members of the software components. These software design decisions make unit testing impossible, because the software components cannot be tested in isolation from the other software components.

In the following, the paper shows how the software architecture for the TET satellite line should change to handle the technical debt in the reused software with slight modifications of the software architecture. To support this, we present the current software architecture applied in several projects, like Eu:CROPIS, which includes changes made due to the lessons learned from FireBIRD and how it effect the test process.

2. SOFTWARE CHANGES IN BIROS

Besides minor software changes in order to increase software robustness in case of malfunctions, major software changes were made due to the requirements of additional experiments on the BIROS satellite. The minor software changes will also be applied to TET-1 after they have been successfully proven in space on the BIROS satellite.

The major software changes are

- a control interface to the nitrogen thruster system for orbit maneuvers
- a 4-quadrant-sensor interface to orient an optical communication system to the laser buoy of the optical ground station (PrOSIRIS) [3],
- extension of the wheel system with three high torque wheels for high agile attitude control [4],
- high speed image reading from star tracker hardware for the optical tracking of a remote target and read out of further information from the star tracker,
- interface for AVANTI [5] to request star tracker data, to access and control the thruster system, and command attitude modes in the AOCS,
- and an extension to the attitude control state machine by new control modes for the experiments.

3. COMPARISION OF SOFTWARE ARCHITECTURES

The software architecture has a big impact of the testability and maintainability of software systems. The design decisions of the AOCS software architecture of the TET line are based on the reused software from the BIRD project. BIRD's architecture decisions were significantly influenced by predefined systems such as the used operating system and the application framework. Besides this, a major factor was the reusability of the software for future space missions. Testability was not recognized as a valid driver for software quality. Therefore, software verification is limited to manually testing the controller functions while driver interfaces have to be tested in a HIL environment. In current projects, the lessons learned from BIRD as well as test automation requirements influence software design decisions.

The TET satellite line uses a component based design. Interactions between the software components are handled by a component manager. It uses a key-value map to return a pointer to the requested software component. Figure 1 shows the basic design of the component-oriented architecture and the process required to setup and access other components. This enables the software components to be replaced with stubs when the stub is registered with the component manager under the key of the stubbed software component. What was not done in the TET line was to define the interface routines of software components as virtual methods in order to allow them to be stubbed.

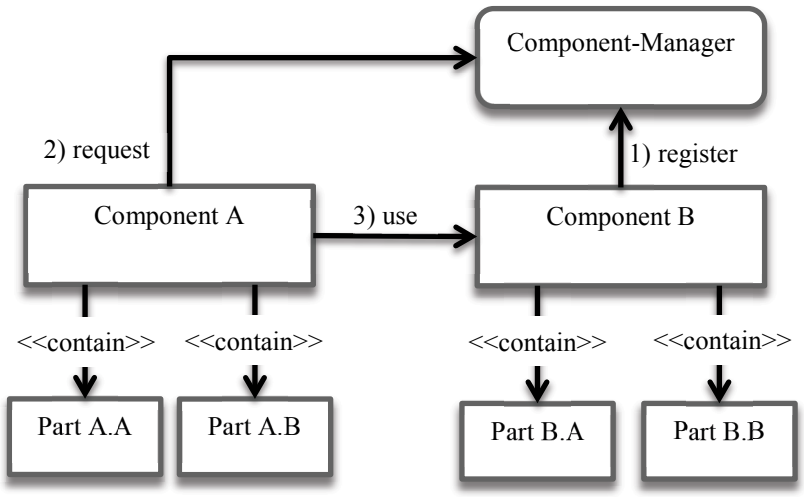

Figure 1 Component design in TET satellite line

The organization of software components in the TET satellite line uses a tree structure. The implemented interface owns the bus interface. This ownership structure makes the stubbing of interfaces in a unit test impossible. Also, the privacy of component data is an obstacle for early tests on software level. With some modifications, it is possible to enable this type of software tests, for example, to break up the ownership to another level in the software architecture and allow stubbing by defining the interface with virtual methods. The only obstacle is multiple inheritances when software components have to provide several interfaces. The requirement of no runtime type information for embedded systems will, in such a case, lead to the wrong method being addressed. Endless loops in thread bodies are another test problem.

In current projects, a data and event flow oriented view of the system is used. In this approach, all data in the system should be part of a channel, so that computational tasks become stateless. The term “software component” is only used in the software architecture to group tasks and their related channels by purpose. The timing in the system is controlled by trigger channels which are connected to a clock to achieve wait operations in a sequence of consecutive tasks or to execute a task periodically. Figure 2 shows the typical setup of the software architecture in current projects.

This software architecture simplifies the setup of unit tests. Also, the used framework provides the functionality to set up and tear down the environment as well as control the clock inside the framework to test the timing of tasks. Therefore, only the instantiation and connection of tasks and channels is necessary inside a test fixture. The test cases have to push the test data into the input channels of a task, start a schedule of the

framework until all connected task have been executed, and finally check the state and data of the outgoing channels against the expected test results.

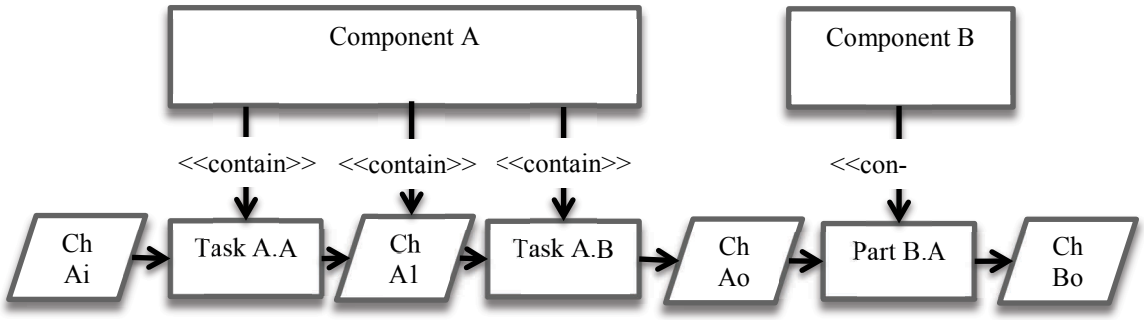

Figure 2 Data flow oriented and reactive design in current projects

4. EXPERIENCES AND CONCLUSIONS

The use of a test-first approach slows down the software development. However, it significantly improves the bug location and fixing process. In the end, the initial commissioning of communication with hardware is speed up. As an example, the ACS sensors in the Eu:CROPIS satellite only required two days. The remaining bugs were related to integration and hardware configuration issues. For TET-1 and BIROS, this development step took several weeks and bug fixing became cumbersome.

In the end, most of the issues in the BIROS software could be fixed during system tests, some during commissioning phase. E.g., the satellite is placed into a controlled spin when the controller falls back to auto acquisition mode when no inertial information is available in an inertial control mode. The reason was a wrong command modification in TET-1 which had no noticeable effect until the fall back was implemented as minor change in BIROS. A huge part of the modifications were related to the star tracker. Here, unit tests can avoid that existing functionality is modified, but also to setup a test case to find the reason for an observable issue in the behavior of the interface software. Nonetheless, we as software engineers know that there will always be one line of code of the 30,000 which will not work for the untested state. Only a good test approach discovers this line, which is activated by a new requirement and its implementation.

5. REFERENCES

[1] H. Reile, E. Lorenz, and T. Terzibaschian. The FireBird Mission - A Scientific Mission for Earth Observation and Hot SpotDetection. Digest of the 9th International Symposium of the International Academy of Astronautics, pp. 17-20 (2013)
[2] O. Maibaum, T. Terzibaschian, C. Raschke, and A. Gerndt. Software Reuse of the BIRD ACS for the TET Satellite Bus. In: Digest of the 8th International Symposium of the International Academy of Astronautics. pp. 409-412. Wissenschaft und Technik Verlag, Berlin (2011)
[3] C. Schmidt, M. Brechtelsbauer, F. Rein, C Fuchs OSIRIS Payload for DLR's BiROS Satellite. International Conference on Space Optical Systems and Applications, Kobe (2014)
[4] C. Raschke, T. Terzibaschian, W. Halle High Agility Demonstration with a New Actuator System by Small Satellite BIROS. 9th Airtec 2014, Frankfurt/Main (2014)
[5] G. Gaias, J.-S. Ardaens Design challenges and safety concept for the AVANTI experiment. Acta Astronautica, vol. 123, pp. 409-419 (2016)