# Agile Software Development for Space Applications

A. Lill[1], D. Messmann, M. Langer

Technical University of Munich
Boltzmannstraße 15, 85748 Garching, Deutschland

**Abstract**

Software development for space applications is characterized by historically grown structures and conservative methods derived from traditional project management. Many of these methods are not easily transferable from normal product development to software development. Project risk is high and delays are the rule due to the many uncertainties regarding the planned cost and time budget, possible requirement changes in later project phases as well as unforeseeable complications. Furthermore, these methods have very limited flexibility and come with highly time-consuming planning, implementation and, if necessary, problem solving.

Agile software development does not require that all requirements are known and well-defined at the beginning of the project. The development is incremental and generates a usable and testable software product with every new iteration. This makes the development more flexible and problems can be detected earlier and solved with less effort. Due to the frequent integration into the existing system, a close collaboration is possible across subsystems as well as the customer or the project partners. This increased flexibility and improved cooperation reduces project risk, cost and time until delivery.

This paper shows the application of agile software development in the space sector applied to a CubeSat project. Within the student satellite project Munich Orbital Verification Experiment II (MOVE-II) at the Technical University of Munich (TUM) the concept of agile software development was successfully applied to develop the software of the on-board computer within a few months. The agile methods presented in this article demonstrate software development that does not require the final requirements at the beginning of the development process. These methods allow that a new version of the software can be tested and operated after every iteration of the process. The launch of our CubeSat MOVE-II is scheduled for early 2018.

## 1. INTRODUCTION

This paper shows the application of agile software development in the space sector, applied to a CubeSat project. It will first give a short introduction into the traditional sequential software development used extensively in the space sector before we briefly describe CubeSats and our own CubeSat project, called MOVE-II. Chapter 2 will focus on sequential software development as well as agile software development and point out key differences. After that the methods and processes from the area of agile software development that we applied to the MOVE-II project will be explained. Following the conclusion and a summary of lessons learned we will provide an outlook for the future of software development in the space sector.

### 1.1. Software Development for Space Applications

For decades, high mission costs forced the space industry to be risk-averse, thus relying on methods, processes and tools with a long heritage. While this approach led to approved, well-known technologies and workflows, software and software developments remained a critical aspect of every space mission. Losses due to software flaws, like Mars Climate Orbiter [1], Ariane 5 flight 501 [2] and Mars Polar Lander [3] are showing the criticality of this process, especially since the complexity of the on board software historically increases with an exponential growth rate of a factor of 10 approximately every 10 years over the last 40 years [4].

Traditionally, software development for space applications adheres to a rigorous, standardized design process. The development methodology usually follows predictive models like the Waterfall model, the Compressed Waterfall Model or the Incremental Build Model. Adaptive Models like the Spiral model or agile software development have been usually used only for less critical software although they promise more flexibility to change and do not require extensively defined requirements [1].

[1] alexander.lill@tum.de, phone: +49 89 289 16017

According to [6], many space programs face the risk of cost growth and schedule delays due to software development issues. A more rigorous software assessment process with earlier detection and correction of software flaws could save more than a third of these costs.

### 1.2. CubeSats

CubeSats [7], standardized satellites that come in multiples of so-called units (10 x 10 x 10 cm), are being developed in universities and companies all over the world. They enable fast and cheap access to space and students to obtain hands-on experience about spacecraft technologies. Thus, novel satellite hardware as well as development processes can be researched with less resources and risk than in traditional space programs.

Within the MOVE-II satellite project [8] of the Technical University of Munich, several novel concepts are investigated. A fault-tolerant, radiation-robust filesystem [9], autonomous Chip Level debugging [10], dependable data storage on miniaturized satellites [11], a novel communication protocol for miniaturized satellites [12] and MicroPython as Application Layer Programming Language [13] have been studied so far.

Results of a statistical analysis of 178 launched CubeSats [14] show that many CubeSats fail due to insufficient testing in flight configuration. Lessons learned of First-MOVE [15] yielded in the goal of developing and testing the software as early as possible in the MOVE-II project. Thus, agile software development was chosen as the software development methodology for MOVE-II.

### 1.3. The MOVE-II CubeSat Project

MOVE-II is a 1 Unit (1U) CubeSat (see Fig. 1) that is being developed by students at the Technical University of Munich (TUM) since April 2015.

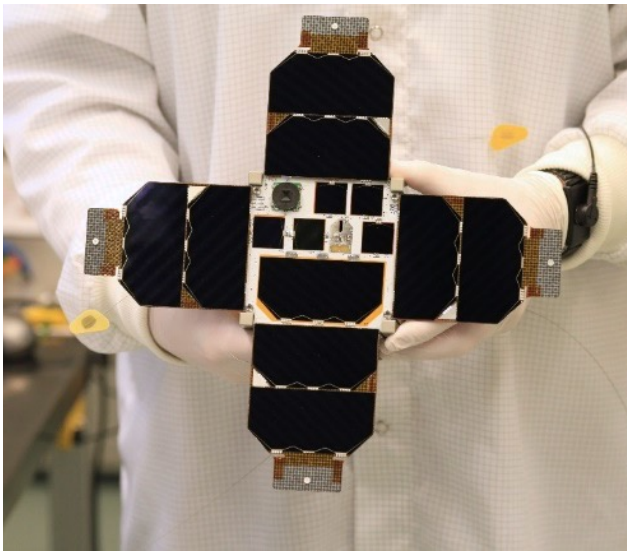

**Fig. 1: MOVE-II in launch configuration.**

The satellite, carrying novel solar cells as a scientific payload [16], will be launched into a 575 km sun-synchronous orbit (SSO) in early 2018.

According to project internal statistics, a total of 102 voluntary students worked over 46,000 hours for the MOVE-II project in a period of one year and ten months so far. As a university project this comes with a high fluctuation in the numbers of active students, weekly working hours and a more complicated transfer of knowledge between the participants. These circumstances also greatly affect the management of the project including the adherence to the project plan and deadlines, causing an even bigger conflict between the three typical forces in project management of using limited resources within a defined time frame to ensure an acceptable quality.

The MOVE-II software architecture is based on a custom distribution of Linux running on our Command and Data Handling Board. This builds the base for many small programs that serve exactly one purpose. Each subsystem is continuously running one of these programs as a so called “daemon”. These daemons provide the interface for each subsystems’ functions and interact with the other daemons over the message bus “D-Bus”. Some programs on the other hand do not need this message bus because they fulfill their purpose once they are started, for example the program that deploys our solar panels and radio antennas by sending commands to our EPS subsystem daemon to activate certain power switches.

Due to the nature of this voluntary project, the limited time and low financial budget and the lower effort for development and testing, we decided to use widely used tools (like Linux, several standard libraries and other commonly used tools as for example the initialization system “systemd”) for the implementation of our system functionality.

## 2. BACKGROUND: SOFTWARE DEVELOPMENT

Software development processes can be grouped into different categories. This section will point out the main characteristics of sequential and agile software development and their key differences.

### 2.1. Sequential Software Development

Sequential software development processes such as the Waterfall model and the V-Model strictly follow planned phases. This starts with the requirements analysis and continues with the phases design, implementation and testing. While the V-Model already uses testing to verify the smallest components, the waterfall model focuses on testing everything once implementation is finished.

This approach works very good when all activities and tasks as well as requirements are well defined and

foreseeable. This process can lead to very problematic situations if unforeseen situations or problems occur, or if these preconditions are not met. These situations often result in being over budget or behind schedule [17].

### 2.2. Agile Software Development

Agile software development processes are always iterative processes where the created artifacts evolve incrementally. The existing result of any iteration can be delivered to the customer and further improved, changed and extended in the next iteration.

Agile software development follows principles as defined in [18]: “Individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, responding to change over following a plan.”

Agile software development is often used in environments where requirements are prone to change. It allows to have a usable product at an early stage of the product lifecycle and therefore allows early testing as well as iteratively delivering improved versions. At the same time the previous version acts as a fallback in case the newest version does not work as intended and enables to deliver a minimum viable product at any given time in the process.

## 3. METHODS AND PROCESSES

During the development of the software for MOVE-II we applied methods and processes that were adapted from the agile software development described in chapter 2. This chapter will provide details about these methods and reasons why we chose to apply them to the MOVE-II project. As the first step the system architecture had to be designed. Afterwards the collection of requirements and the implementation of the minimum viable product was conducted. The created artifacts were then tested as well as iteratively improved and extended.

### 3.1. Subsystem Decomposition

The software operating on our satellite can be split into different groups using their purpose or the hardware they are interacting with. Following this approach, we grouped all software components into the already existing subsystems of the satellite, namely: Structure (solar panel and antenna deployment software), Thermal (temperature monitoring software), Attitude Determination and Control Software, Communications software, Electrical Power System (electrical power control and analysis software), Payload software and Command and Data Handling software which also provides common functionality for all other software.

This enabled us to create clearly defined interfaces between the different software components and subsystems and allowed us to develop independently until we integrated the software on the satellite. Every subsystem daemon has the exclusive right to speak to its hardware, e.g. EPS to the EPS board using an Inter-Integrated Circuit ($I^2C$) bus or ADCS to the ADCS board using a Serial Peripheral Interface (SPI) bus.

### 3.2. Collection of Requirements

Following the subsystem decomposition, the requirements for every software component had to be defined. For this, we created a high-level description of all functionality and interfaces for every component. These descriptions included external functions for every component as well as functions that all components must have, as well as details about the internal functionality of the components. These internal requirements reduced the possibilities of how the requirements can be implemented and further simplified the process of creating the first minimum viable product.

Following the agile software development approach these requirements were then successively specified in more detail to reduce the number of possible interpretations as well as changed and adjusted if necessary. Overall the most basic interfaces and definitions did not change much as the most common changes where clarifications and a higher level of detail.

### 3.3. Minimum Viable Product & Focus on System Tests

After the collection of requirements, the first goal was to implement a very first version of all the software components in a few days. The so-created version constituted the Minimum Viable Product of our software and therefore included the most important functionality and interfaces and allowed us to run it together with the other components on the hardware. This enabled us to detect problems that occurred in interaction with other subsystems and their software at an early stage and quickly find solutions for those problems.

### 3.4. Iterative Improvement and Continuous Delivery

After the implementation of the Minimum Viable Product there has always been an artifact that could be tested and incrementally changed. This means that the basic functionality can be improved and extended, and that additional features can be added.

Using automatic build pipelines and the version control system “git” and the web interface “GitLab”, every change to our software triggered the delivery of a new artifact that we could deploy to our hardware and test together with the rest of the system. In case the new

feature did not work, we were able to roll back to the last version that was used and continue development with this successfully tested artifact. Incorporating agile methods like “Scrum” the software development was organized into weekly sprints for which the tasks were prioritized, estimated and then committed to by the team of developers once a week. Thus, every week brought a new version of our software components that we deployed to the satellite and tested.

Furthermore, all changes to our stable and already tested version were reviewed by another developer before they were incorporated into the existing version. This ensured a high code quality and a broader distribution of knowledge about changes, known issues, best practices and the progress in the team.

## 4. CONCLUSION & OUTLOOK

We presented an example of agile software development methods and processes applied to a space application, the MOVE-II CubeSat project. We successfully developed all software for our satellite in a timeframe of roughly 9 months. Having a Minimum Viable Product of our software components after two months enabled us to detect hardware issues and start system level testing early on.

Compared with the traditional software development for space applications where purely sequential processes are used we could integrate all hardware and software as soon as all hardware was available and find mistakes in hardware and software design and implementation and fix those issues without causing budget overruns or being behind schedule.

We believe that this approach can positively influence other projects where the system tests might otherwise be neglected due to insufficient time near the end of the project. Also, non-CubeSat projects can use this knowledge to further improve their processes and maybe incorporate more agility in their missions.

## ACKNOWLEDGMENTS

The authors acknowledge the funding of MOVE-II by the Federal Ministry of Economics and Energy (BMWi), following a decision of the German Bundestag, via the German Aerospace Center (DLR) with funding grant number 50 RM 1509.

## REFERENCES

[1] JPL Special Review Board. “Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions”.In: (March 2000).

[2] J.L. Lions. “Ariane 5 Flight 501 Report of the Inquiry Board”. In: (July 1996).

[3] E.E. Euler et. al. “The Failures of the Mars Climate Orbiter and Mars Polar Lander: A Perspective from the People Involved”. In: Proceedings of Guidance and Control 2001 paper AAS 01-074 (2001).

[4] Daniel Dvorak. “NASA study on flight software complexity”. In: AIAA Infotech@ Aerospace Conference and AIAA Unmanned Unlimited Conference, 6 - 9April 2009.

[5] C. Krupiarz, A. Mirantes, D. Reid, A. Hill, and R. Ward: “Flight Software”, in: *The international handbook of space technology*, M. Macdonald and V. Badescu, Springer, 2014.

[6] J. Eckardt, T. Davis, R. Stern, C. Wong, R. Marymee and A. Bedjanian, “The Path to Software Cost Control”, in; Defense AT&L, November-December 2014.

[7] H. Heidt, J. Puig-Suari, A. Moore, S. Nakasuka, and R. Twiggs, “CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation,” Proceedings of the AIAA/USU Conference on Small Satellites, 2000.

[8] Langer M., Appel N., Dziura M., Fuchs C., Günzel P., Gutsmiedl J., Losekamm M., Meßmann D., Trinitis C.: “MOVE-II – der zweite Kleinsatellit der Technischen Universität München“, Deutscher Luft- und Raumfahrtkongress 2015, Rostock, Germany, September 22-24 2015.

[9] Fuchs C. et. al. “A Fault-Tolerant Radiation-Robust Filesystem for Space Use”. In: Lecture Notes in Computer Science Volume 9017 (2015), pp. 96–107.

[10] Fuchs C. et. al. “Enhancing Nanosatellite Dependability Through Autonomous Chip-Level Debug Capabilities”. In: ARCS 2016; 29th International Conference on Architecture of Computing Systems (2016), pp. 1–4.

[11] Fuchs C. et. al. “Enabling Dependable Data Storage for Miniaturized Satellites”. In: Proceedings of the AIAA/USU Conference on Small Satellites, Student Competition, SSC15-VIII-6 (2015).

[12] N. Appel et. al. “Nanolink: A Robust and Efficient Protocol for Small Satellite Radio Links”. In: Proceedings of the Small Satellites Systems and Services – The 4S Symposium 2016 (2016).

[13] Plamauer, Sebastian, and Martin Langer. "Evaluation of MicroPython as Application Layer Programming Language on CubeSats." ARCS 2017; 30th International Conference on Architecture of Computing Systems; Proceedings of, VDE, 2017.

[14] M. Langer, J. Bouwmeester, "Reliability of CubeSats – Statistical Data, Developers' Beliefs and the Way Forward", Proceedings of the 30th Annual AIAA/USU Conference on Small Satellites, Logan, UT, 6-11 August, 2016, Paper SSC16-X-2.

[15] M. Langer, C. Olthoff, J. Harder, C. Fuchs, M. Dziura, A. Hoehn, U. Walter: "Results and lessons learned from the CubeSat mission First-MOVE", in: Small Satellite Missions for Earth Observation, R. Sandau, H.-P. Roeser und A. Valenzuela, Springer Berlin Heidelberg, 2015

[16] M. Rutzinger, L. Krempel, M. Salzberger, M. Buchner, A. Höhn, M. Kellner, K. Janzer, C. G. Zimmermann, M. Langer, "On-Orbit Verification of Space Solar Cells on the CubeSat MOVE-II", 43rd Photovoltaic Specialist Conference (PVSC), IEEE, Portland, Oregon, June 5 - June 10, 2016.

[17] J. Moffat, "Software Engineering in the Aerospace Industry – Why is it so hard", CSCI 5828, 2010.

[18] Kent Beck, James Grenning, Robert C. Martin, Mike Beedle, Jim Highsmith, Steve Mellor, Arie van Bennekum, Andrew Hunt, Ken Schwaber, Alistair Cockburn, Ron Jeffries, Jeff Sutherland, Ward Cunningham, Jon Kern, Dave Thomas, Martin Fowler, Brian Marick, "Manifesto for Agile Software Development". Agile Alliance, 2010.