www.DeepakPublishing.com

Karvinen, K., et al. (2015): JoSS, Vol. 4, No. 1, pp. 301–314
(Peer-reviewed Article available at www.jossonline.com)

www. JoSSonline.com

Using Hobby Prototyping Boards and Commercial-off-the-shelf (COTS) Components for Developing Low-cost, Fast-delivery Satellite Subsystems

Kimmo Karvinen, Tuomas Tikka, and Jaan Praks

Aalto University School of Electrical Engineering, Espoo, Finland

Abstract

The use of small satellites has increased significantly in recent years, and a growing community is participating in designing and building space instruments. To achieve lower costs and faster delivery times, and to allow innovative solutions from people with multidisciplinary backgrounds, easy and straightforward development processes and tools are needed. The possibility of having people involved from different areas of expertise than solely space engineering could bring out innovation and approaches that would not otherwise be possible.

This paper presents a process for developing low-cost, fast-delivery satellite subsystems using commercial off- the- shelf (COTS) components and investigates the suitability of a readily available open source hobby development board (Arduino) in the prototyping phase of this process. The process allows participation of external groups to the project with a low threshold. The external group can be a group of experts from different fields, or even a group of students or hobbyists. This practice can facilitate innovation, and lower risks from design uncertainties in the beginning of development. Open source hobby prototyping tools with COTS components have several potential benefits for developing small satellite subsystems: low component and platform prices and shorter development times, achieved by using a platform that takes care of most low-level issues, saves time in the beginning of the prototyping process, and allows earlier subsystem and satellite level verification.

The current study involved the implementation of sun sensor subsystems for Aalto University's nanosatellites using the tools and processes presented herein, and investigated whether the prototyping process and platform is suitable for the task. An included use case goes through the sun sensor subsystem development process, along with specifications needed to build one. The process was successful, and the final subsystem requires an external envelope of only 6 mm * 6 mm * 1 mm for integrating its sensor part, allowing easy integration to solar panels and being one of the smallest and cheapest satellite sun sensors available.

Corresponding Author: Kimmo Karvinen, kimmo.karvinen@iki.fi

1. Introduction

The use of small satellites has increased significantly in the past years, especially nanosatellites weighing only 1–10 kilograms (Swartwout, 2013). The lower cost compared to conventional satellites allows universities, small start-up companies and non-space-faring nations to develop experimental spacecraft and subsystems (Woellert, 2010). Also, the traditional space industry has moved towards a faster, better, cheaper (FBC) approach (Paxton, 2007).

The central idea of building nanosatellites is generally to achieve faster delivery and lower costs compared to conventional satellites, typically at the expense of reliability and performance. To keep costs low, using commercial of the shelf (COTS) components and subsystems is very typical (Dubos, 2010). Many mass-production and open-market electronic components and integrated circuits (IC) may be utilized in small satellite systems, even if they are designed for terrestrial use. They often offer sufficient performance for a nanosatellite with a much more affordable price than space-grade equivalents. However, dedicated methods and processes are required for validating their durability and performance early within a satellite project, since their suitability to fulfill mission requirements is often unclear.

Many nanosatellite subsystems can be procured from the market and integrated to the satellite with ease, if for example CubeSat Kit specifications are followed. The availability of such subsystems has further decreased development times of small satellite projects, allowing further overall project cost and time reductions. However, some subsystems required for a specific satellite mission may not be available off-the-shelf, or do not offer the best possible compatibility with the rest of the satellite design. Thus, they have to be developed concurrently within a fast-paced nanosatellite project.

This paper examines the suitability of using an open-source hobby prototyping platform (Arduino) in a new refined development process of building nanosatellite subsystems by using COTS components, while presenting Aalto University's nanosatellite project as a case example. Using such a platform in the early phase of the development would potentially allow validating component selections and provide valuable input to satellite level design. Also, since a large number of nanosatellite projects have educational objectives, a simple, easily available and open-source prototyping platform would potentially make the development project more approachable for a wider group of people and lead to interdisciplinary innovations.

A development project of a miniature digital sun sensor subsystem is presented as an example, but the same practices can also be used in the development of other small satellite subsystems built with COTS components. The sun sensor subsystem was developed for the needs of the electro-static plasma brake/sail experiment onboard the Aalto-1 nanosatellite (Khurshid, 2014) and as the main absolute attitude sensor in the Aalto-2 nanosatellite, taking part in the international QB50 atmospheric science mission (Gill, 2013).

This paper is divided into eight sections: The second section presents related work and background on the subject. The third section presents a development process for small satellite subsystem development using COTS components. The fourth turns to discussion of the potential benefits of using hobby development platforms, and the fifth section presents a use case of sun sensor subsystem prototyping. The sixth section presents the steps taken after the prototyping phase for the final space-instrument development, and the seventh presents results and discussion, with concluding remarks in section eight.

2. Small Satellite Design Methodology

To permit timely and reliable subsystem development concurrently within a fast-paced small satellite project, new development and quality processes should be investigated. Unfortunately, traditional space industry standards cannot be followed directly, and there is little public information available about actual practices used in small satellite subsystem development and testing. Still, even in the case of nanosatellites, considerable but low-cost efforts should be taken to validate systems under development. In many nanosatellite projects, inadequate development practices have caused too little time and

effort for thorough software development and interface testing, which may lead to project delays, increased costs, and high infant mortality rates (Swartwout, 2013). Fortunately, it is expected that a properly tailored combination of traditional parts assurance techniques and assembly level screening with qualification testing can also produce a reliable design for nanosatellites (Rose, 2012).

Small satellite projects are usually very dynamic by nature, and requirements may change late in the project due to many ambiguities caused by the concurrent design approach. To control the associated risks, agile system engineering practices generally used in software projects have been proposed to be used in nanosatellite projects (Huang, 2012). Moreover, new verification strategies have been developed (Eickhoff, 2007) (Hendrics, 2005), allowing concurrent development and earlier satellite level verification. In these strategies, simulation and development models are used to allow starting software development as soon as possible, and verifying interfaces between different subsystems before the final hardware is available. To obtain realistic analysis results from such simulation environments, subsystem characteristics should be validated and included in the simulations.

As many small satellites are used either for educational or technology demonstration purposes (Swartwout, 2013) and the project teams consist of a small internal group of space technology professionals, the importance of allowing the use of new innovative technologies and external experts should be emphasized. The typical organizational structure of nanosatellite projects at Aalto University is depicted in Figure 1 as an example. In a small satellite organization, the internal group is formed of space technology professionals responsible for project planning, specifications, management, and final implementation, while external groups are interdisciplinary contributors participating only on a defined part of the project.

Measuring increased innovation potential is not unequivocal, as are component cost and development time. Nonetheless, innovation research supports the idea of accelerating innovation by using a greater number of internal and external sources for ideas. Effective use of both internal and external paths leads to better results than relying solely to internal experts (Chesbrough, 2006). One of the main parts is to integrate internal and external knowledge with a combination of more complex knowledge (Chesbrough, 2006). Modular small satellite projects provide a

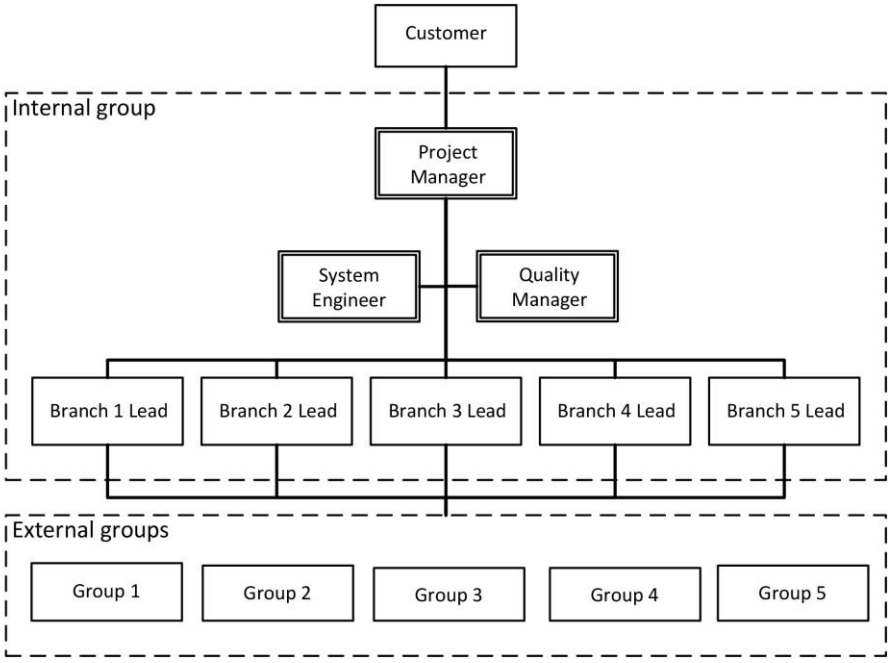

Figure 1. Small satellite project organizational structure.

good setting for this kind of approach, as long as there is a straightforward process in which different groups can participate. Providing an approachable process and development platform to build satellite subsystems enables others, in addition to space technology professionals, to contribute in a way that is directly adaptable to the actual project.

3. Small Satellite Subsystem Development Process

An agile process for developing small satellite subsystems using COTS components and external groups is presented in this section. The process is refined from the traditional space system development process flow depicted, e.g., in the ECSS-M-ST-10C (European Cooperation for Space Standardization, 2009), but takes into consideration the need for agile development and early validation of subsystem designs. The main differences to the standard process include an iterative requirement specification approach, further emphasis on testing already at a prototyping phase, and a straightforward interface for including external developer groups. The development process used in Aalto University's nanosatellite projects is presented in Figure 2, showing inputs to the concurrent satellite level design process.

The development starts by setting out requirements on the satellite level, which fulfill the mission statement and intended functions. Satellite-level requirements form the baseline for development, and are further complemented with lower level requirements, typically for each subsystem or development

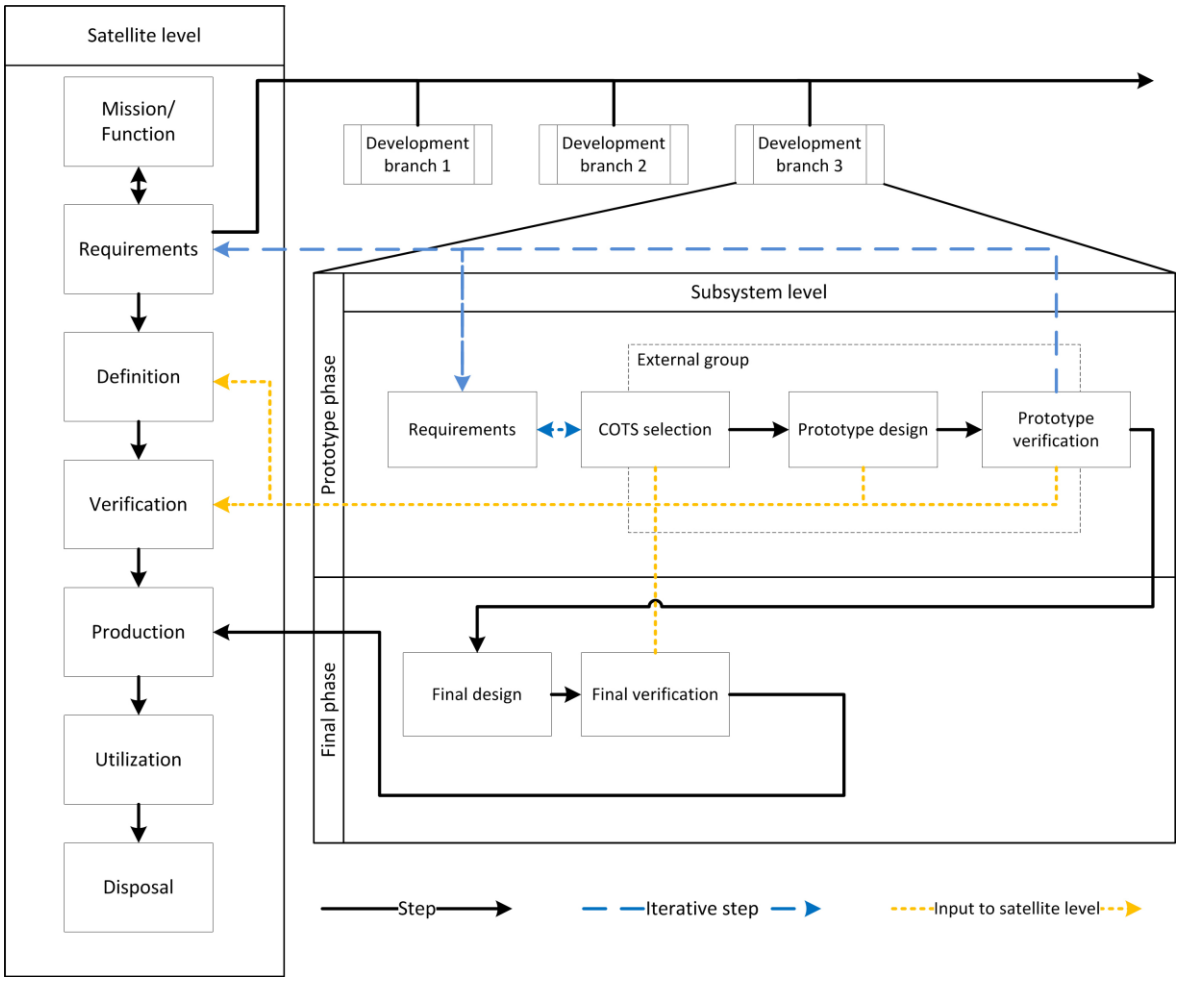

Figure 2. COTS subsystem development flow with iterative requirement specification, input to satellite level design and an option of including external groups.

branch. A single developer can be contributing to several development branches, and aspects of a single subsystem may be part of several development branches.

While satellite level requirements are given as an input, exact subsystem requirements and design values are iterated on-the-fly, during prototype development and testing in meetings between affected internal and external groups. These updates can be added to requirement compliance matrices, automated design budgets, and satellite level analyses. One should note that the subsystem level requirements can be decreased only if the satellite-level system can compensate for it. Typical key requirement types and sources from which they can typically be determined, even in the very early part of the project, are presented in Table 1 as an example.

Table 1. Key Requirements and Sources from Which they are Defined

Requirement/ Subsystem level	Source / Satellite level
Mass	Mass budget
Power	Power budget
Dimensions	CAD model
Interface	System diagram
Performance	Mission simulation / analysis
Environmental durability	Standard / analysis

Since very strict requirement specification limits the possible outcomes of the prototyping process, a certain degree of freedom should be allowed. Very strict requirements assure the applicability in the main project, but also diminish the possibility for innovative and unconventional solutions. One should take into account that making specifications that cannot be fulfilled by using any commercially available options is a possibility. Hence, the specifications need to be open for modifications according to COTS suitability and availability. As soon as the component candidates have been selected, preliminary characteristics from datasheets can be included in the satellite-level design tools, e.g., simulators and budgets, to investigate their suitability for the mission.

Each development branch goes through the prototyping phase and final phase independently. The prototyping phase is used to identify applicable COTS technologies, test them, and confirm that the subsystem-level requirements are applicable with available COTS components. External groups may be used in this phase for COTS selection, prototype design, and to some extent for prototype verification, as shown in Figure 2. Proceeding to the final phase in a development branch means freezing down the subsystem requirements and building a flight equivalent system.

The verification of the subsystem is divided into prototype and final verification for early assessment of the selected COTS components in relation to the specified requirements. Once the prototype version is ready, already in the very early part of the development, a testing campaign is conducted. The testing may follow traditional commercial electrical, electronic, and electromechanical (EEE) component tests, depicted, e.g., in ECSS-Q-ST-60-13C (European Cooperation for Space Standardization, 2013), tailored to meet the required level of reliability. One should note that the durability and performance of the procured components may differ from components tested by the manufacturer as they are usually not from the same manufacturing batch and may have been modified. Especially if environmental durability requirements exceed the tests done by the manufacturer, the components should be tested with appropriate levels already in the prototyping phase. These tests usually include further thermal tests and radiation testing. Also, subsystem interfaces should be tested with functional tests on other subsystems or subsystem simulators.

4. Using Open-source Hobby Development Platforms

A readily available development platform for the prototyping phase, taking care of most low-level issues, such as connecting input and outputs along with an integrated development environment (IDE), would allow a wide range of developers to be involved in the subsystem design project. Arduino Uno is currently the most popular open-source hobby development board, including a microcontroller, a boot-

loader, high level C++ libraries and an integrated development environment (IDE) (Arduino Uno, 2014). It has a well-established user base and wide support from both online communities and component suppliers. David Cuartielles, the co-founder of Arduino, stated in 2013 that there were over 700,000 registered official Arduino boards. In addition, he estimated that there is at least one clone board per every official Arduino (Medea, 2013).

While members of external groups can be experts in their own area, it is possible that they are not embedded system professionals. Advantages of a straightforward prototyping processes are highlighted with interdisciplinary groups and non-engineers. However, for non-engineers, even the threshold of getting started with building a simple prototype is very high without approachable tools.

When the main functionality is conducted with Arduino, there is usually a need for optimization on component price, mass and volume. Arduino Uno uses an Atmega328P microcontroller (Atmel Atmega328P Datasheet, 2014), which comes in many packages, making optimization easier as it can be used in many final products. Usually this can be done without bulky support electronics used in the development board itself. This also eliminates the need for reprogramming the software after the prototyping phase.

Software created by the process described in this paper includes parts of code created by the external group, as well as open source parts such as code examples and libraries. Some advantages of the open source code are quite obvious, for example the possibility to freely use work done by others and promoting stronger user involvement. The open source movement has also shown its strength by creating some well-established and widely used software, e.g., Linux, Apache and Mozilla.

As with any external software, the quality of the code should be checked. According to studies, open source projects have some features that support accelerated software development as well as patterns that lower the overall code quality. More people looking at the code allows more “bugs” to be found, which leads to faster software improvement. (Quality Assurance under the Open Source Development Model p 3) However, this also leads many developers to rely on users to validate their software and to publish minimally tested software (Zhao, 2003).

It has been found that user participation in open source projects can be very high, and users discover 20% to 40% of the faults in 20% of the projects. This kind of high user activity is achieved generally in large scale projects, while smaller projects cannot expect much contribution. (Zhao, 2003)

Stamelos, et al. (2002) measured quality characteristics of 100 open source Linux applications by using Logiscope software measurement tool. These results showed the structural code quality is actually higher than could be expected from a limited control development process, but still lower than the quality implied by the standard proposed by the tool itself. According to the used tool, nearly half of the components of each application examined called for revisiting the code.

According to the results cited above, software quality assurance should be emphasized when using open source components. Using external groups that consist of people who are not experts on programming may lead to a need for revisiting and rewriting parts, or even all of the code after the prototyping phase. Comparing these issues with closed source software is impossible, because it cannot be read to estimate possible errors. These results should not be misinterpreted that using closed source would solve code quality issues. Moreover, even if there would be a suspected bug, with closed source, one must wait and rely on the manufacturer to fix it. With open source, it is possible to examine every part of the code and verify its suitability for the intended use.

The following sections investigate the suitability of Arduino Uno as a prototyping platform for COTS components in the prototyping phase of the development process presented in the previous section.

5. Use Case: Sun Sensor Subsystem Prototype Development

In the case of the sun sensor subsystem, requirements caused by the rest of the Aalto-1 nanosatellite design prevented using any commercial sun sensor subsystems from the market. The external

envelope of the sensor part had to be small enough to fit on all sides of the satellite for omnidirectional measurements, without causing the need to remove any solar cells. An attitude and orbit dynamics simulator, satellite level requirements, and analysis by the electro-static plasma brake payload and ADCS providers were used to determine requirements for the sensor. The key requirements for the subsystem in the beginning of the development project are shown in Table 2.

Table 2. Sun Sensor Subsystem Key Requirements

Requirement	Value
Mass	30 g
Power	30 mW
Dimensions	6 mm * 6 mm * 6 mm (external)
Interface	I^2C
Performance	5 deg accuracy (1σ) in 90 deg FOV when Sun visible
Environmental durability	Temperature range: –70 – +100°C
	Radiation tolerance: 100 krad total dose

These requirements were given to an external group, who proposed Elmos E910.86 Integrated Solar Angle Sensor (Elmos Semiconductor, 2010), shown in Figure 3, to be used for prototyping with Arduino Uno. Before moving to prototyping design, COTS component selection was approved by the internal group, as incompatible component selection at this point would render the next phase useless. This section describes the prototype design process done by the external group to prototype verification, which is done in cooperation with the internal group, as was shown in Figure 2.

Commonly, sensors are connected to hobby boards using breadboard or pin headers. However, like many COTS integrated circuits, the selected sensor E910.86 is physically too small to be used with Arduino in this manner, as it is meant for surface mounting. Thus, a protoboard, shown in Figure 4, was manufactured for the E910.86 with needed resistors, capacitors, and a pin header. A mounting socket for the Quad Flat No-leads (QFN) package component could also be used. However, in our case it would have complicated the performing tests for the prototype.

E910.86 uses Serial Peripheral Interface (SPI) to communicate with Arduino. SPI.h library is included to handle conversations, as defined in the SPI protocol (SPI library, 2014). SPI is loosely defined, and some values, which are predefined in other protocols, must be specified. SPI clock polarity and clock phase must be set to 0, which defines timing for all conversations between Arduino and E910.86. This means the data is read on the rising edge and

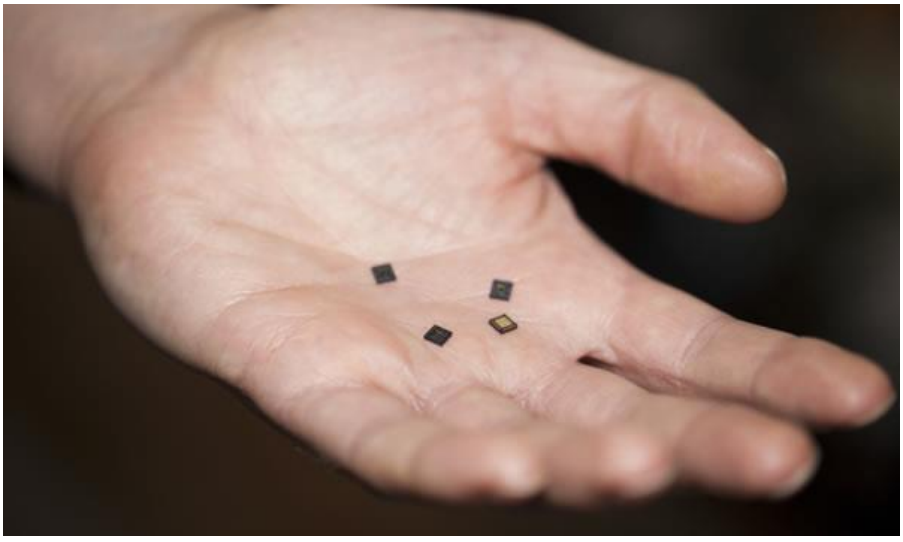

Figure 3. Elmos E910.86 solar angle sensors sized 4 mm * 4mm * 0,5 mm on hand for scale.

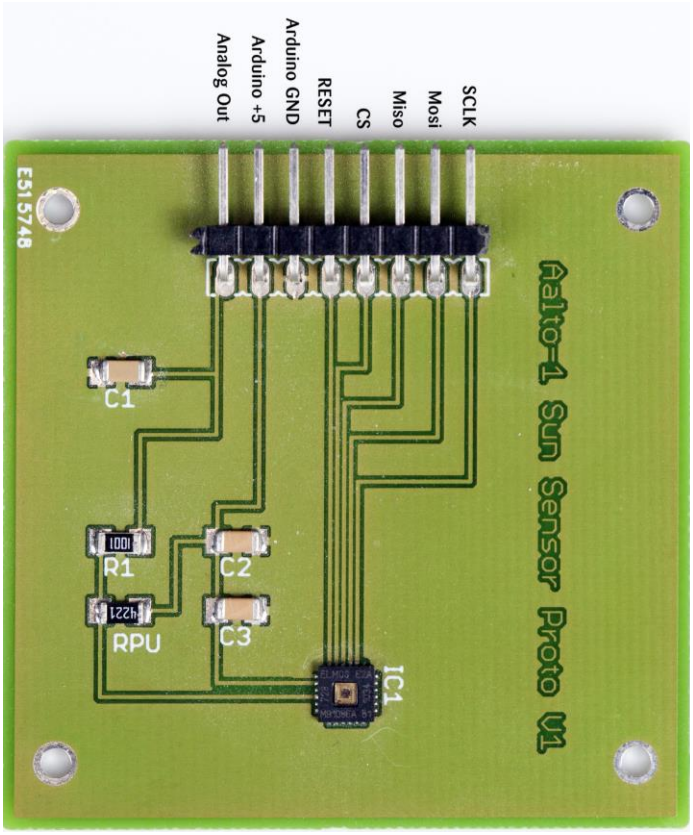

Figure 4. Sun sensor subsystem prototype version 1.

propagated on the falling edge. Bytesex must be set to the most significant byte first. SPI's clock pulse frequency needs to be set to 1/4 of Arduino Uno's speed. Detailed description of the software has been published online (Satellite Sun Sensor Prototype Tutorial).

Before starting the loop, the "master" sends two bytes defined in E910.86 data sheet to enable the sensor, and starts the conversation by pulling SS-pin low, and sends two bytes of data to ask xy-angle from the "slave." The slave answers in two 8-bit-pieces, which are combined into one 16-bit-value. This data must be parsed to get usable x and y values. Bit shifting is used to collect the four first bits to verify that package is correct type. Then, bit shifting and bit masking is used to pick bits, which are defined in the data sheet to form a raw value. The raw value can then be converted to degrees. After this, the same procedure is done for y values (Satellite Sun Sensor Prototype Tutorial).

When the sun sensor subsystem is connected to the satellite, the onboard attitude determination and control system (ADCS) (Tikka, 2013) will ask a sun vector from it. ADCS is the master and the sun sensor subsystem is the slave. The E910.86 is reading values all the time, but outputs them via I2C only when it receives a query from the master. As the solar angle sensor uses SPI and the ADCS I2C, the Arduino was programmed to use two different protocols and to provide an interface between them.

As soon as the prototype was working, interface and performance tests were conducted. Since the ADCS was not yet available when the sun sensor subsystem prototyping was started, we used another Arduino Uno to simulate the ADCS, as depicted in the block diagram Figure 5.

Using two Arduinos instead of one revealed a challenge for debugging: Arduino IDE has a built-in support for reading serial port over USB, but it only works for one device. This could have been solved by

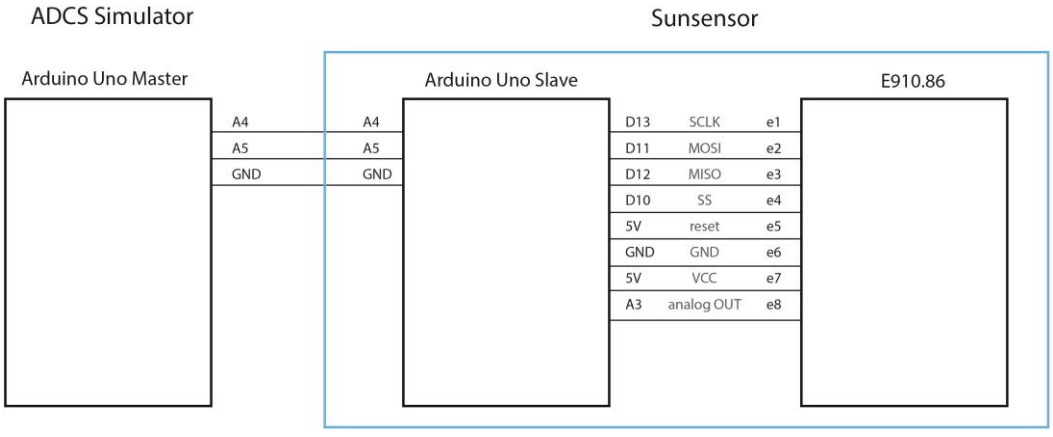

Figure 5. Block diagram of sun sensor subsystem connected to ADCS simulator.

using separate computers for master and slave, but the order and timing of the messages would not have been seen. In addition, it would have made the basic setup much more complicated. To be able to use just one computer, a Python program was made to read two serial ports. This simple program opens two serial ports, reads both from the serial buffer, and prints their messages in different colors. Furthermore, to automate the code compiling and testing for both Arduinos, a Makefile (for GNU Make) was created to compile and upload both codes, and run python code for reading master and slave serials.

The E910.86 sensor's accuracy was specified to be 5 degrees in its datasheet, but was tested during the prototyping phase to verify it would be sufficient for the scientific goals in conjunction with other attitude sensors onboard the satellite. The accuracy tests were performed on a high accuracy optical table, using a motorized rotation stage (Thorlabs CR1/M-Z7) with wobble of less than 2 arcsecs, and a wide spectrum xenon light source simulating the Sun. This preliminary performance testing showed the E910.86's accuracy is highly quantized, as shown in Figure 6, due to a total of 56 raw angle output states in its 150 degree field of view. Even though some larger errors occur momentarily, the sensor's standard deviation (1σ) was calculated to be 3 degrees in the required 90 degree field of view.

In addition to light angle measurements, the sensor was programmed to provide light intensity and temperature data. The light intensity information is used to remove light angle measurements coming from light sources other than from the Sun, most importantly from Earth's albedo. During the mission, the ADCS selects a sensor with the highest intensity value, if it is over a certain threshold. The intensity threshold is calibrated on ground with a high power sun simulator, but can still be adjusted in orbit if needed. The temperature data is on the other hand used in sensor calibration, as all temperature dependencies can be removed. The validated operational characteristics of the sensor were modelled to an attitude and orbit dynamics simulator in Simulink for testing the whole ADCS control loops. The sensor models can also be used in hardware-in-loop (HIL) testing of the ADCS control algorithms running on real hardware (Tikka, 2013).

An environmental test campaign was conducted for the prototype version. The external placement of the sensor on the satellite surface causes thermal

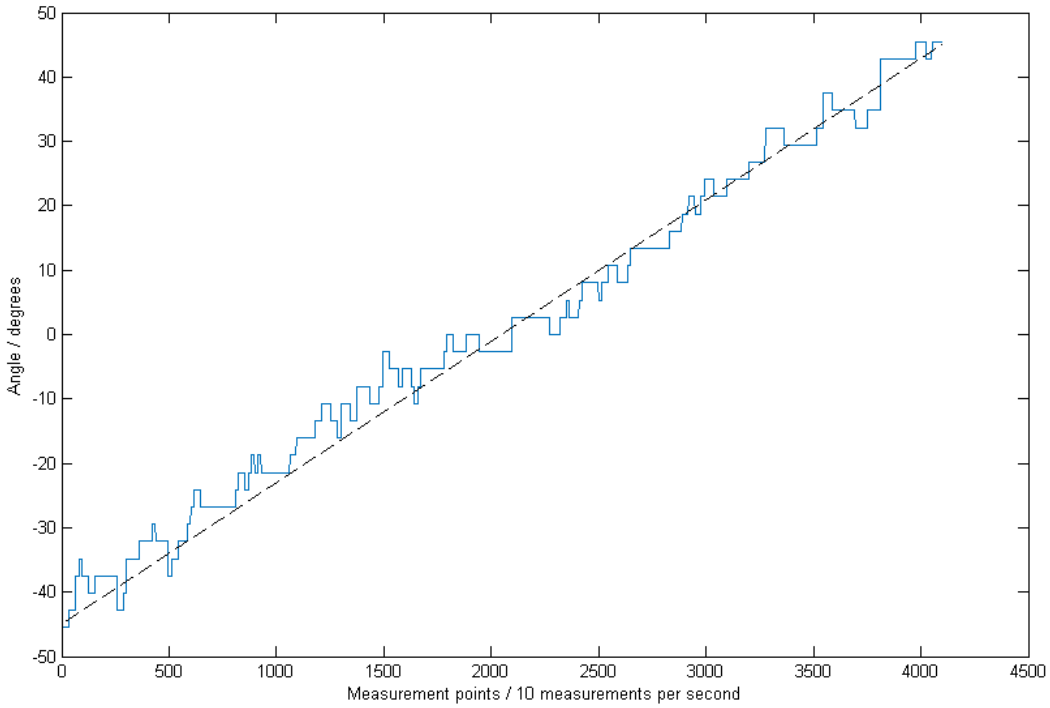

Figure 6. Sun sensor measurement values from –45 deg to 45 deg. True values shown with linear line.

variations that are larger than the component design values –40 to +85°C. Thus, the sensor was thermal-cycled while operational for one week from –70 to +100°C, according to analysis and environmental test standards, and showed to operate without degradation even in extreme temperatures. Also, radiation testing with 50 MeV protons was performed to investigate if single events occur. No single events were noticed; however, the E910.86 stopped working after an estimated 11 krad total dose. Since the total dose experienced by the sensor at the satellite exterior in duration of the Aalto-1 mission was expected to be much higher, analysis was performed to investigate whether adding a cover glass on top of the solar angle sensor would provide the required tolerance.

The results from the tests at prototyping phase allowed validating the sensor's key parameters and were decided to fulfill the given requirements, if a cover glass is added over the sensor part. Accuracy tests were performed again with a cover glass, and showed no effect on the accuracy of the sensor. The tests also provided a good confidence measurement of the sensors operation in the space environment. Thus, the final qualification campaign for the sun sensor subsystem can be performed later in the project without major risks, when the final durability and performance requirements are frozen and the rest of the ADCS hardware is available.

6. Use Case: Towards the Final Space Instrument

After the prototype phase was finished, the performance and durability of the sensor had already been verified preliminarily. The system had been shown to fulfill all key requirements, and a decision was made to move forward to final development. Two separate versions were designed; a stand-alone and a solar panel (Finnholm, 2012), integrated versions shown in Figure 7. For both versions, an integrated microcontroller and other necessary components were added to the bottom side of the PCB. The final product has an external envelope of only 6 mm * 6 mm * 2 mm and the total dimensions of the stand-alone version is 18 mm * 18 mm * 6 mm.

The sun sensor subsystem design was decided upon, to exploit the prototype development as much as possible. The same microcontroller as in Arduino Uno, the Atmega328P in Quad Flat Package (QFP), was used in the final version for providing the I2C interface and including all measurement interpretation and calibration as a stand-alone system. This also allowed reusing the software written in the prototyping phase, applying sufficient software quality assurance techniques discussed earlier.

A PCB design was made to include all necessary components for software uploads using an AVRISP mkII programmer shown in Figure 8. Finally, a

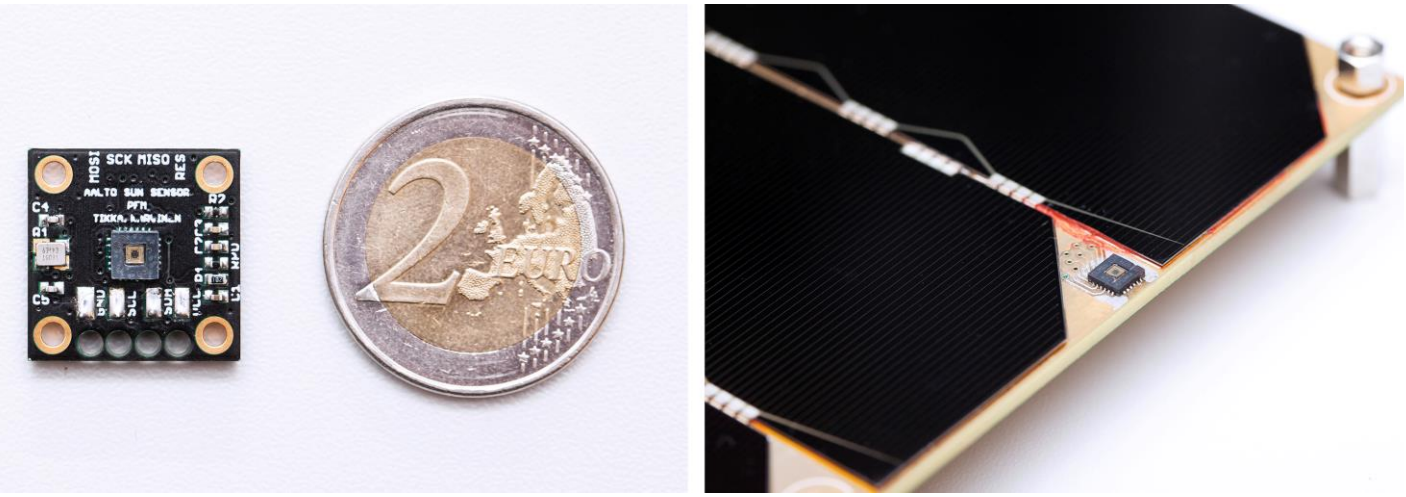

Figure 7. Stand-alone sun sensor subsystem and sensor integrated on a 3U CubeSat 8 cell solar panel.

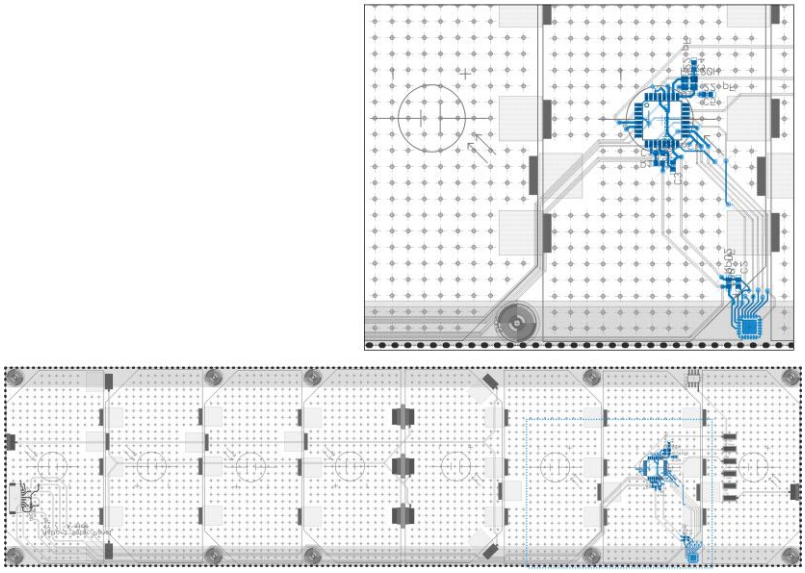

Figure 8. PCB design of sun sensor subsystem integrated on a 3U CubeSat 8 cell solar panel.

commercial high-reliability connector was included in the design, for robust connectivity to the satellite main communication bus.

A final testing and calibration campaign, including the same tests as for the prototype model, was performed for the final versions to verify their operation in the flight equivalent configuration. Additionally, further tests are naturally performed in the satellite level qualification and acceptance campaigns. In the end, the final development phase was finished with relatively little additional workload and good confidence of the sensor's suitability for the mission, due to the possibility of using a large amount of work from the prototype development and tests. The final characteristics of the developed sun sensor subsystem are depicted in Table 3.

Table 3. Sun Sensor Subsystem Final Characteristics

Requirement	Value
Mass	10 g
Power	8 mW
Dimensions	6 mm * 6 mm * 2 mm (external)
Interface	I^2C
Performance	3 deg accuracy (1σ) in 90 deg FOV
Environmental durability	Temperature range: –70 – +100°C
	Radiation tolerance: 100 krad total dose

7. Results and Discussion

A COTS subsystem development process was proposed in Section 3 and followed in the development of the Sun sensor subsystem. It was noted that by performing prototype development and testing with Arduino Uno, all necessary subsystem characteristics can be easily investigated preliminarily, giving valuable input to component selection and concurrent satellite level design. Also, since the final space-instrument was implemented on the same hardware platform as Arduino Uno, the software development and subsystem testing were able to begin, before the final subsystem hardware was implemented. These factors could potentially lower risks associated with nanosatellites due to the suitability for a concurrent design project, and earlier interface and environmental testing possibilities.

According to this experience, choosing components is the biggest factor in defining needed time and technical skill level for this kind of prototyping project. If the size and package of the component allows it to be directly connected to Arduino or a breadboard, some steps that are not beginner friendly can be skipped. The need for circuit design and sur-

face mounting clearly raise the required skill level for prototyping, making it less ideal for non-engineer builders. People using hobby boards, such as Arduino, usually work with components built for easy connecting, which are documented and often have code examples.

In addition to connecting components, available libraries and code examples define how easy the implementation is. For people who are proficient with embedded systems design, it does not make such a big difference, but for less technically orientated groups, examples may be crucial. E910.86 uses SPI, a protocol that is loosely defined. Lack of strict definitions makes using generic SPI codes impossible. For example, the polarity and phase of the clock signal, word length, and bytesex are chosen by the component manufacturer. There was no reference implementation for the E910.86, and writing the SPI interface for Arduino based on its datasheet took over one week of work. With this protocol, there is a high threshold to get even a simple response from the component as a small error renders it completely nonfunctional. For comparison, an unrelated SPI component that had not been used before, a HMC5983 magnetometer from Honeywell, was tested. By using Arduino examples, it was working in less than 10 minutes.

The setup for using two Arduinos and simulating the satellite ADCS is also out of the scope of the most ordinary Arduino use, and would be challenging for people with limited programming experience. Providing Arduinos simulating the subsystem interfaces to groups building prototypes would remove one obstacle from the prototyping process, and ensure that the prototype functionality is compatible with the rest of the satellite design. Arduino would be a sensible option, as it is affordable and its operation is relatively easy to understand. Such sample subsystems would also make more comprehensive testing possible in very early stages of the project.

One of the biggest advantages of using Arduino or other popular hobby development platforms is the online community support and ready-made codes with instructions for the components. Satellite component requirements may nullify this advantage, as the needed off-the-shelf components might not be used by hobbyists. One way to prevent this would be to use widely used components for the first version of the prototype. This strategy would be more feasible in a more complex part than the sun sensor subsystem, which has only two main components.

Using external groups consisting of people with limited programming experience underlines the need for software quality assurance techniques. All external code should be reviewed and the necessary parts rewritten after the prototyping phase. Also, if COTS components are replaced after prototyping, for example with space qualified models, a similar process is needed. The possible need for changing the components does not however obliterate the advantages of the presented prototyping process, such as multidisciplinary accessibility, rapid development, and early concept verification.

8. Conclusions

The development process used successfully combined external work with the main project, gave valuable input to satellite level development, and the created subsystem fulfills the requirements given to it. It is one of the smallest sun sensor subsystems available, and the accuracy is sufficient for the missions for which it was developed. The code package, including the sun sensor subsystem and the ADCS simulator, was published online (Satellite Sun Sensor Prototype Tutorial, 2014).

Despite the strong hobby and do-it-yourself (DIY) background, Arduino proved to be a viable development platform for satellite part prototyping. It solves many of the low level requirements; however, the process required solutions that would have been challenging for participants with less technical background. The use case findings imply that COTS component selection has a surprisingly large impact on prototype building difficulty level. This should be taken into account when using external groups without advanced embedded system skills. In some cases, the internal group could help with the component selections or even make pre-selections.

The cost of the sun sensor subsystem was relatively low. The development of eight sensor subsystems (six for flight and two for development models)

cost less than 1,000 € in total. The calculated price includes the component and manufacturing prices, and excludes personnel and test equipment costs. The actual project cost depends largely on the workforce used and developer's test facilities. After testing and building the first patch of sensors, the development and personnel costs for the following pieces would be considerably lower. The cost of six commercially available alternatives would have been in range of 15,000 – 59,400 € in total (CubeSatShop). Commercial alternatives available on 12.2.2015 included CubeSat Sun Sensor (2,500.00 €), SSOC-A60 2-Axis accurate sun sensor (4,890.00 €), SSOC-D60 2-Axis accurate sun sensor (7,890.00 €), and Digital Fine Sun Sensor (9,900.00 €) (CubeSatShop).

The current test project, which was limited to just one subsystem, cannot verify the possible advantages of using external groups for increasing innovation and knowledge. Nevertheless, the process enabled earlier subsystem verification and using diverse groups with different expertise for satellite subsystem design and prototyping, with some restrictions and prerequisites. It is in any case clear that new development processes and practices, refined from current industry standards, would benefit small satellite development and should be investigated further. In the future, it would also be interesting to test this with external groups using loose requirement specifications for a part, given the possibility for more innovative and surprising approaches. Interdisciplinary groups could also be used to simultaneously provide different solutions for the same requirements.

Acknowledgments

The authors would like to thank Tero Karvinen, Ville Kyrki, Hannu Leppinen, the Multidisciplinary Insitute of Digitalisation and Energy, and Tekes – the Finnish Funding Agency for Innovation.

References

Arduino FAQ (David Cuartielles) (2013): Available: http://medea.mah.se/2013/04/arduino-faq/ (accessed May 5, 2015).

Arduino Uno (2014): Available: http://arduino.cc/en/Main/arduinoBoardUno (accessed May 5, 2015).

Atmel Atmega328P Datasheet (2014): Available: http://www.atmel.com/Images/doc8161.pdf (accessed May 5, 2015).

Chesbrough, H. (2006): *Open Innovation: The New Imperative for Creating and Profiting from Technology*. Boston, Harvard Business School Press.

CubeSatShop (2015): Available: http://www.cubesatshop.com/ (accessed February 25, 2015).

Dubos, G., et al. (2010): Statistical Reliability Analysis of Satellites by Mass Category: Does Spacecraft Size Matter? *Acta Astronautica*, Vol. 67, pp. 584–595.

European Cooperation for Space Standardization (2009): Space Project Management, Project planning and implementation. Available: http://www.ecss.nl (accessed February 25, 2015).

European Cooperation for Space Standardization (2013): Space Project Assurance, Electrical, Electronic and Electromechanical (EEE) Components. Available: http://www.ecss.nl (accessed February 25, 2015).

Eickhoff, J., et al. (2007): Model-based Design and Verification – State of the Art from Galileo Constellation Down to Small University Satellites, *Acta Astronautica*, Vol. 61, pp. 383–390.

Elmos Semiconductor E910.86 Datasheet (2010): Available: www.mouser.com/ds/2/594/910_86-224506.pdf (accessed May 5, 2015).

Finnholm, J., et al. (2013): Design and Manufacturing of Aalto-1 Solar Panels, in *Proc. of the 2nd IAA Conf. on University Satellite Missions and Cubesat Workshop*, pp. 726–734.

Gill, E., et al. (2013): Formation Flying Within a Constellation of Nano-satellites: The QB50 Mission. *Acta Astronautica*, Vol. 82, Issue 1, pp. 110–117.

Hendricks, R. and Eickhoff, J. (2005): The Significant Role of Simulation in Satellite Development and Verification. *Aerospace Sci. and Technology*, Vol. 9, pp. 273–283.

Huang, P., et al. (2012): Agile Hardware and Software System Engineering for Innovation, presented at the IEEE Aerospace Conf., Big Sky, MT, March 3–10, 2012.

Khurshid, O. and Tikka, T., et al. (2014): Accommodating the Plasma Brake Experiment On-board the Aalto-1 Satellite, in *Proc. of the Estonian Academy of Sciences*, Vol. 63 (2S), pp. 258–266.

Paxton, L. (2007): "Faster, Better, and Cheaper" at NASA: Lessons Learned in Managing and Accepting Risk. *Acta Astronautica*, Vol. 61, Issue 10, pp. 954–963.

Rose, R., et al. (2012): CubeSats to NanoSats; Bridging the Gap Between Educational Tools and Science Workhorses, presented at the IEEE Aerospace Conf., Big Sky, MT, March 3–10, 2012.

Satellite Sun Sensor Prototype Tutorial (2014): Available: http://botbook.com/satellite/ (accessed February 25, 2015).

SPI library (2014): Available: http://arduino.cc/en/Reference/SPI (accessed May 5, 2015).

Stamelos, I., et al. (2002): Code Quality Analysis in Open Source Software Development. *Inform. Syst. J.*, Vol. 12, Issue 1, pp. 43–60.

Swartwout, M. (2013): The First One Hundred CubeSats: A Statistical Look. *JoSS*, Vol. 2, pp. 213–233.

Tikka T., et al. (2013): Low-cost and Fast-delivery Verification Strategy for the Aalto-1 Nanosatellite Attitude Determination and Control System, presented at the 5th Nano-Satellite Symposium, University of Tokyo, Japan, November 2013.

Tikka, T., et al. (2013): Attitude Determination and Control System Implementation for the Aalto-1 Nanosatellite, in *Proc. of the 2nd IAA Conf. on University Satellite Missions and CubeSat Workshop*, pp. 676–694.

Woellert, K. (2011): Cubesats: Cost-effective Science and Technology Platforms for Emerging and Developing Nations. *Advances in Space Research*, Vol. 47, pp. 663–684.

Zhao, L. and Elbaum, S. (2003): Quality Assurance under the Open Source Development Model. *J. of Syst. and Software*, Vol. 66 (1), pp. 65–75.