70th International Astronautical Congress (IAC), Washington D.C., United States, 21-25 October 2019.

IAC-19-D1.3.4

Software Systems Engineering: The Underlying Infrastructure for Enabling and Repurposing Space Systems

Martin Ristov[a], Dr. Elliott Coleshill[b], Kin Wing Tsui[c], Rohaan Ahmed[d]

[a] *Department of Software Engineering, MacDonald, Dettwiler and Associates, 9445 Airport Road, Brampton, Ontario, Canada L6S 0B6,* martin.ristov@mdacorporation.com
[b] *Yeates School of Graduate Studies, Ryerson University, 350 Victoria Street, Toronto, Ontario, Canada M5B 2K3,* elliott.coleshill@gmail.com
[c] *Department of Software Engineering, MacDonald, Dettwiler and Associates, 9445 Airport Road, Brampton, Ontario, Canada L6S 0B6,* kin-wing.tsui@mdacorporation.com
[d] *Mission Systems Department, MacDonald, Dettwiler and Associates, 9445 Airport Road, Brampton, Ontario, Canada L6S 0B6,* rohaan.ahmed@mdacorporation.com

Abstract

Software is an integral piece of all modern spacecraft design. In the present day, it supports mission- critical operations such as rendezvous and docking, autonomous navigation of rovers, mid-course adjustments, attitude control, re-entry and managing of spacecraft's systems. The first manned spacecraft – Mercury, was not equipped with a computer. In fact, NASA spent its first fifteen years running Earth- orbital and deep space missions without an on-board general-purpose computer. As the natural evolution of more complex missions came about, technology rapidly advanced in parallel, serving as the supporting force behind the enablement of more sophisticated onboard computers. This progression allowed for enhanced capabilities, which has made possible for spacecraft to be more intelligent, increasing the value of the software component. Examples are reflected in the ever-growing functionality and autonomy of spacecraft, as well as the volume of mission data that is collected and pre-processed in-orbit.

Expansion in intelligence is the result of the software's ability to change the functions of the computer in which it resides, and by extension, the hardware it controls. As missions constantly transform to best fit their desired result, using software to adjust for the changes is much more cost-effective than replacing or even altering the hardware, which in some cases is not even an option. Software also provides a means to recover from various malfunctions, and compensate for hardware defects and failures – potentially extending mission life.

This paper explores the importance and role of software in space systems, focusing on a new software enhancement, currently in development, to automate, and later add artificial intelligence to the Mobile Servicing System on-board the International Space Station. With deep space exploration and the newly planned Lunar Gateway, our next generation space robotics will have to operate without human intervention, using artificial intelligence to maneuver. Additionally, this paper explores past achievements in which software had a role in recovery, repurposing and even extending mission life, such as fixing an operation-critical anomaly of the MSS robotic grappling hands (Latching End Effectors), extending the life of gyros on-board Earth and Mars-orbit bound spacecraft, tuning the traction control algorithm, adding autonomous navigation, and repurposing driving sensors into gravimeters on the Mars Curiosity rover.

Keywords: *software, architecture, repurposing, reconfiguring, autonomous, spacecraft, systems*

Acronyms/Abbreviations

BC - Bus Controller
CFS – core Flight System
CPU - Central Processing Unit
CSA – Canadian Space Agency
FPGA - Field Programmable Gate Array
ISS – International Space Station
MAST - Modular Autonomous Systems Technology
MIL-STD – Military Standard
MSS – Mobile Servicing System
NASA - National Aeronautics and Space Administration
SIMD - Single Instruction-Multiple Data
V&V - Verification and Validation

Introduction

Flight software is a type of embedded software, a field that has seen exponential growth since its inception. This growth trend is expected to continue due to the increasingly ambitious requirements, the advantages of situating new functionality in software or firmware rather than hardware, as well as the ability to address issue in-flight. This has made flight software to become a spacecraft's "complexity sponge", because it readily accommodates evolving understanding, making it an enabler of progress. Whether it be a satellite or a rover, adjustments and upgrades to the subsystems' software are a common occurrence on almost any mission. These updates help adjust the spacecraft to its environment by

calibrating sensors and fine-tuning hardware, which in turn extends mission life, and in some cases even increase science return.

Sections:

Acknowledgements
John Beaulieu, MDA Interim Software Engineering Manager, for his continued support.

1. Computer Software in Spacecraft Design

Software allows us to build systems with a level of complexity and coupling that is beyond our ability to control. In fact, we are building systems where the interactions among the components, often controlled by reconfigurable software, cannot all be planned, understood, anticipated, or guarded against. This change is not solely the result of using digital components, but it is made possible because of the flexibility of software [1].

In today's spacecrafts, it is used for guidance and navigation functions such as rendezvous, re-entry, and mid-course corrections, as well as for system management functions, data formatting, and attitude control [1]. It enables highly complex activities - satellite precision manoeuvres, autonomous navigation of rovers on a planetary terrain, preprocessed scientific analysis – as well providing a means to recover from malfunctions, and even compensate for hardware defects.

Software must work correctly first time in orbit, and getting it right is crucial to mission success, as software bugs have been the doom of many spacecraft, going back to the earliest days of space exploration. While this may make it seem like a double-edged sword, the added value of software to space systems has greatly increased in recent decades, reflecting the ever growing functionality and autonomy of spacecraft, and the amount of mission data to be collected and processed [2].

2. Spacecraft Re-Configuration

2.1 Software updates in active space-missions

A space mission is considered successful once the spacecraft completes all of its planned scientific objectives. Depending on the type of mission, once the spacecraft remains operational well beyond its expected mission life, it is considered as even a bigger success as the science return will only continue growing.

Whether it be a satellite or a rover, adjustments and upgrades to the subsystems' software are a common occurrence on almost any mission. These updates help adjust the spacecraft to its environment by calibrating sensors and fine-tuning hardware, which in turn extends mission life, and in some cases as illustrated in the following sections, even increase science return.

2.1.1 Re-purposing components to increase science output

A notable software upgrade enabled Curiosity to immediately send high-priority images to Earth, instead of waiting to send them at day's end. In late summer of 2018, NASA engineers uplinked another round of software to the rover that gave Curiosity the ability to do some of its own navigation during drives across the Martian surface. Previously, Curiosity took pictures of the surrounding terrain and sent them back to NASA, where the images were studied and the rover's next drive was meticulously plotted.

The rover also carries accelerometers, normally used for navigation and attitude determination. An innovative software upgrade recalibrated them to isolate the signature of the changing gravitational acceleration as the rover climbed through Gale crater [3]. This made for a completely new set of science to be run, not originally planned in the rover's science instrument kit design.

2.1.2 Increase of autonomy

The Canadian Space Agency (CSA) has proposed adding an autonomous capability to the Mobile Servicing System (MSS) to benefit current ISS robotic operations. This upgrade will reduce the number of MSS robotics commands that need to be sent simplifying operational procedures and planning by providing a common library of basic operation-scripts. These scripts would be used as building blocks for

various operations, ultimately enabling autonomous, as well as vision-guided control of the MSS manipulators.

2.2 System architecture considerations

Since updates occur on almost every space mission, whether it is to enhance capability or work around failures of hardware, a distributed software architecture would address various pain-points across most design and operational stages. Not only would this allow for a component-based design, debug, testing and integration, it streamlines how software updates are planned, developed, tested and uplinked.

In this section, a few system architectural considerations are discussed. The main drivers behind it are the abovementioned processes, as well as adding the infrastructural foundation and flexibility for an artificial intelligence layer to be later introduced, should the hardware and mission objective support it.

From the initial design of the software concept of operations, the understanding of what data will be needed for the functionalities that are desired is essential. Data selection will drive sensor design, selection, number, and placement. Management of data will dictate storage and processing architecture and hardware. Adopting design methodologies that encourage a model- and data-centric view of the systems need are key, as it would ensure that the appropriate hardware architecture is available to support the software and autonomy needs of the system [4].

2.2.1 Data communication

Using a data communication bus architecture such as MIL-STD-1553B is a good candidate for carrying data between the various on-board electronic components. Other than reducing cost and weight, it provides benefits and determinism to ensure predictable behavior.

- Multi-channel coordination and selection – the Bus Controller (BC) would effectively provide the means to segregate specific commands intended to reach recipient system components. Therefore, these commands will be carried out in a guaranteed order of priority.
- Real-time redundancy – dual redundancy entails the activation of a redundant backup bus controller upon failure of the primary one.
- Efficiency - data burst and transmission rates of 1 MHz would allow for a more flexible window in regards to response time from other subsystems [5].
- Synchronization – a bus based architecture provides a common clock for all distributed components.

Communication between components of the spacecraft system should ideally use the same protocol. This allows for single-purpose systems to communicate to one or more components simultaneously.

All incoming and outgoing packet structures should be pre-defined, which the flight software will use as a reference to dynamically generate Command/Telemetry packets, based on loaded configuration files. In the case of an artificial intelligence-powered system, internal parameters such as previously loaded neural states and training parameters should be stored, logged and reloadable.

Most, if not all, of the decisions for system and subsystem monitoring and control are based on the actual values in the operational data. Portions of the decisional control will rely on configurable data to interpret or act on the operational data. As a result, the executable software becomes an “engine” whose operation is largely dependent on and driven by the value(s) of the various types of data it handles. Building a configurable data system in this way allows for the modification and validation of data while minimizing flight software modifications [4].

2.2.2 FPGA vs CPU

Spacecraft data processing applications that demand high throughput are often streaming processes such as signal and image processing, instrument calibration, and feedback control loops. These are well-suited to Single Instruction-Multiple Data (SIMD) computational models. Most reconfigurable computing implementations perform best on these kinds of problems, which fit naturally into a dataflow computing model [6].

When deciding FPGA, CPU or both, the following things should be taken in consideration:

- Processing Time
- Utilization Requirements
- Power Requirements
- Thermal Requirements
- Data rates/clock speeds

A hybrid solution, part FPGA part CPU would be suited for this type of system - utilizing an FPGA for hardware interfacing, buffering and high speed processing, and a CPU for lower speed data transmission and processing.

2.2.3 Goal-based configuration

As illustrated in the previous examples of software updates to the Curiosity rover, the purpose of the updates greatly varied, each with a specific goal in mind. Extending the life of the rover’s wheels effectively extended the rover’s life. Upgrades to the navigation algorithm and accelerometers allowed the rover to gather more science. In deep space missions where spacecraft operations have communication gaps with the ground, these goals would be fed into the “intelligent” system, which would in turn autonomously reconfigure the different spacecraft components to best satisfy the given mission priority. Further details on this concept are covered in subsequent sections of this paper.

2.2.4 Ease and accuracy of testing

Due to the functions that remain active during a communication blackout are critical, it is essential that any autonomous functionality be sufficiently tested and guaranteed before deployment. The assurances required of these autonomous functionalities to gain the trust of ground controllers constitute the validation and verification (V&V) of these systems. The challenge becomes that the system will be deployed in largely unknown and unanticipated conditions. As such, the ability to fully test the system, particularly software systems that employ learning technologies or model adaptations, would have to be developed. New ways of testing and monitoring systems for the satisfaction of their intended functionality, both prior to launch and during deployment, are needed.

Another challenge is that the verifications must be given, in many cases, in a nearly absolute sense, such as declarations that the system can "never" fail. Demonstrating assume-guarantee contracts, runtime monitoring, and some formal method approaches are all potential candidates towards solutions to this V&V problem [4].

3. Autonomous Software Reconfiguration

In deep-space missions, the environment is unpredictable and constantly changing. In an ideal scenario, continuous updates for various adjustments would be highly desirable - be it for navigation, anomaly correction, compensation, or performing science experiments.

In terms of deep-space missions however, the spacecraft would have very limited, if any contact with ground, directly delegating this role to software with decision-making capabilities.

In the context of repurposing a space system, a modular software architecture to centralize, monitor and manage all hardware and sensors telemetry would be required. The intelligent feature in this software system would give the on-board computer the ability to analyze states, through inputs like sensor telemetry and vision, and act upon these various scenarios.

As such, solutions that increase the autonomy of the spacecraft should respect both the independence and interconnectedness of the spacecraft subsystems [7]. This distributed and hierarchical approach to system monitoring and control is a key consideration to keep in mind when designing a fully autonomous system, and is something the developers of NASA's core Flight System (cFS) have taken into consideration.

Autonomous system design must consider several things with respect to increasing the time to criticality for any failure, namely robustness, failure strategies, and architecture for critical avionics fail-over or repair [4].

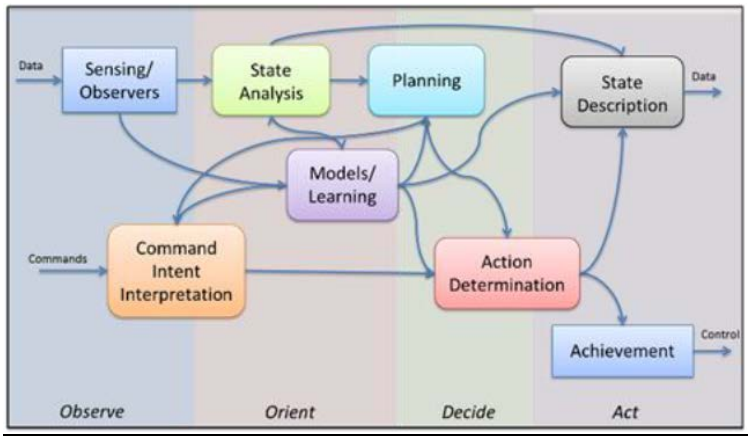

Figure 1: Open-loop Framework Diagram, A Distributed Hierarchical Framework for Autonomous Spacecraft Control [1]

3.1 Autonomy drivers

A NASA analysis on Spacecraft Dormancy Autonomy for a crewed mission to Mars [4], identified the following autonomy drivers, which have the most influence on the need for spacecraft autonomy: time to criticality and the reduced situational awareness due to latency and reduced communication bandwidth. Time to criticality is related to time to effect, it is the time between first sign of a problem and that problem critically affecting operations in the absence of human intervention, whether that be crew or ground. Time to criticality can be lengthened by system design and the incorporation of autonomous functions. In future deep space missions, more data collection, storage, prioritization, processing, and analysis will need to happen without humans in the loop. As such, the constraint that will be placed on access to data drives much of the need for autonomy. An onboard vehicle system manager, capable of cross-vehicle state assessment, fault response, planning, and commanding, is therefore a major need.

3.2 Autonomous fault detection, isolation and recovery (FDIR)

In deep space missions, significantly longer loss of communication scenarios and time delays are expected. This implies that the "obtaining a better, more desired state" needs to be addressed onboard. In the event of a fault, assuming redundant avionics hardware such as a cold or hot spare computer, network switch or mass storage is available, the first response is to switch to the redundant system. A simple autonomy to recover from software faults, such as an autonomous reboot of the component to see if it can be revived and a handback could occur. If the component cannot be reactivated into an acceptable state, the redundant component would remain in control until which time the ground could

intervene and restore the failed component to an operational state.

Since this ability is highly dependent on the avionics architecture, further research is needed to determine the best distributed algorithms for determining when a fault occurs, where it occurs, and which hardware has the command authority to recover the fault. This entails that the architecture also needs to make sure a failure in one component does not take out one or more other components. Additionally, the system would require the fail-over capability to autonomously determine if one system is alive, requires communication, electrical connections are intact and handle scenarios that internal communication is not completely disabled.
Event detection and the subsequent categorization may use an existing model, a learned model, or a combination of both. However as mentioned earlier, further research is needed to assure that event impacts on structural integrity are understood and no events are missed. It is also required that the algorithms minimize or exclude false positives, with the goal to avoid the cost of the action plan that results from a mischaracterization of an event.

Supplemental to the event detection, the autonomous system health manager would require the ability to localize, correctly assess and interpret local events. For example, identifying serious threats versus benign events. Localization is something that currently requires human input, either by looking at the data or by crew inspection. In order to increase the amount of time that can pass between human observation of the system, the autonomous system should be able to deduce a determination of where the event occurred in order to decide on next steps as needed. Furthermore, in a crewed mission, autonomous assessment would provide the crew members or ground controllers with better and faster threat information, which could be invaluable in case of a life or mission critical threat [4].

3.3 In-situ sensor network data analysis
Increasing the robustness of the sensor network would fill the final gap for an autonomous space system. However, the process of determining causation, localization, and structural assessment from a sensor network's operational data is a difficult task, given the complexity of the structure, the possible amount of data, the environmental unknowns, and the relative lack of situational awareness. The autonomous system health manager would then require the development of low and high fidelity structural static and dynamic models, as well as the system's response models to represent the actual hardware. These models would be at the heart of the functional autonomous assessment and characterization algorithms. That said, these models would obviously require the development and ground truth testing to insure their outputted values, including the assessment of baseline conditions. Additionally, predictive algorithms which are able to assess damage states resulting from impacts and system failures will need to be further researched and developed. The development and implementation of advanced learning systems may also be required to satisfy long-term requirements. The spacecraft's autonomous systems must be able to make decisions using the processing capabilities resident on board, even while such decisions involve solving complex problems with many inputs [4].

3.4 Learning and adapting
As more autonomy is designed into space systems, ensuring robustness of the communication system is critical not only for sending up commands and getting telemetry back, but also to learn how the vehicle responds to the on-board autonomy and what needs to be improved upon.
Learning and adapting will be important for the autonomous functions on uncrewed spacecraft, particularly since it will operate in an unknown environment. Currently, significant human time investment is required to collect and annotate data required for learning [4]. As such, technologies must be developed to enable the autonomous annotation and classification of data for learning systems. While this seems like a circular problem, promising work has been explored in this area, particularly in the field of classifying Earth Observation imagery, as well as real-time object detection and annotation in self-driving car technologies.

3.5 Smart telemetry
Because of the long delays in ground receiving telemetry, the implications are that stored telemetry would subsequently be made available, so that an appropriate action or change could be assessed.

Considering the potential for excessively large volumes of stored telemetry that would need to be sorted through if everything were collected, an intelligent system that adjusts the telemetry collection based upon the onboard states and changes would be ideal. Autonomous FDIR powered by a long-term and short-term memory manager would cover desired system behavior in order to remain operational, but the additional insight provided by the adjustments could assist in future exploration vehicles' avionics design. While there is currently a good amount of disentanglement approaches of machine learning algorithms, further research for an application in this scope is needed.

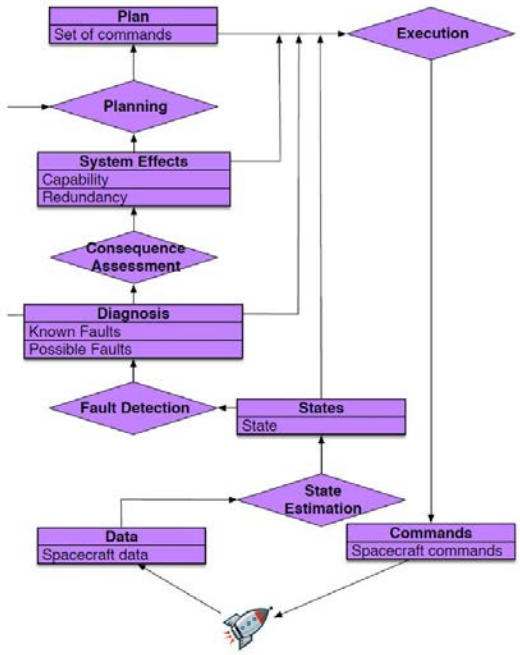

Figure 2: Autonomous Functions, Spacecraft Dormancy Autonomy Analysis for a Crewed Martian Mission

When procedures that consist of configuration changes and control actions are being autonomously executed, not only will expected data need to be available to the procedure execution engine, but associated events that are unexpected must also be identified, captured, and readily available for analysis. Because these interactions are by definition unexpected, the data itself must be collected in such a way that possible associations will be explored. This may require annotations, logging and storage methods, and deep analysis functions in order to capture the unexpected consequences [4].

3.6 Scripted execution

Because of significant time delays, critical transition activities will require some degree of autonomy, somewhat dependent upon the pace of desired transition in conjunction with the time latency. Failures in the state change procedure may require an autonomous recovery option, such as going back to the initial state until the issue can be remedied by the ground.

For operations that must occur due to short time to criticalities, workarounds or autonomous functions to get to the new state would be necessary. Monitoring and logging script execution in support of state transitions will be necessary in order to determine the successful execution of steps associated with the transition as well as verification that the final transition has been successful [4].

3.7 Redundancy and criticality

In terms of classifying the autonomous manager's software, it would fit in the same category as the rest of the flight software. This entails that it can be disrupted or disabled by the same problems that can impact flight software: radiation induced hardware faults, computer faults, loss of power, or software defects, either executable or configuration errors.

Due to this, redundancy for the autonomous software would be accomplished by replicating the software itself, either as multiple processes within a single computer or across multiple computers. This would provide protection against many of the critical failures that can disrupt or disable the software altogether.

More complex protections could include storing the backups in different parts of the spacecraft [4], or even in onboard, mobile robots . These protection mechanisms have significant implications on the avionics architecture, specifically on the data and command pathways to compute assets that typically are not used for these purposes.

4. Applications of Artificial Intelligence and Machine Learning

4.1 Lunar Gateway robotics

The autonomous control software on the Gateway external robotics will integrate with the Gateway Modular Autonomous Systems Technology (MAST) framework [7], which distributes decision-making function through a system hierarchy such that autonomous functionalities are exercised at the lowest level of the system abstraction capable of restoring and maintaining its own most-functional state. On the International Space Station (ISS) the two primary control authorities for robotics are onboard crew and ground operators.

In deep-space scenarios, including in lunar orbit, where ground communication bandwidth is crew presence is expected to be limited, an onboard autonomous vehicle systems management scheme is required for long-term maintenance and control of the station. Gateway robotics, along with other vehicle modules and vehicle-level systems, must be adapted from the design of its low-earth-orbit predecessors to accommodate supervision and control by an autonomous control authority without precluding ground or crew control during planned or off-nominal activities.

As an autonomous agent within the MAST framework, the external robotics system coordinates with the Gateway-level vehicle system manager to define a high-level goal, plans the most appropriate sequence of tasks to achieve the goal, executes the tasks, and re-plans as unexpected situations are encountered. At the level in

the system hierarchy the contextual information required for the decision making during the performance of the tasks are most accessible – without direct involvement from crew or ground operators. Likewise, in fault-detection scenarios where sensor telemetry indicates a failure on the robotics system, isolation of, and recovery from, the fault that caused the failure is executed by the robotics system manager onboard the Gateway, in place of the ground-based engineering analysis process typical of ISS operations.

Key to autonomous fault recovery is the contextual data associated with the system where the failure is detected and isolated, the method of failure detection and its level of uncertainty, as well as concurrent vehicle-level task execution and the states of the vehicle and the other subsystems. The robotics system will be required to understand this context for fault isolation and recovery, deferring to a higher-level autonomous system where lack of contextual information prevents complete recovery.

While autonomy in the MAST framework implies independence from direct ground control, the nominal role that ground operations will assume in the overall mission has abstracted to a goal-oriented programming. Reducing the scope of low-level operation planning and scheduling to off-nominal situations where the highest level of autonomous system still lacks sufficient context to resolve known anomalies. Whereas earlier generations of station external robotics rely on ground support for mission and engineering operations for the detailed planning, scheduling, execution, and fault handling, much of these functionalities will be carried out by the onboard autonomous system, changing the paradigm of ground operation from one with largely an engineering emphasis to one focusing on mission-level goals and plans.

With scheduling, execution, and fault handling functions managed by the onboard system, ground operator contributions will be through defining goals and scheduling execution. Ground support is still required for certain fault handling, as certain faults may require knowledge unavailable onboard for isolation and recovery, but the overall scope of the involvement is reduced. The transition of ground operations paradigm from engineering operations to mission planning is a gradual process in proportion to the level of trust that system operators gain through the execution of tasks and plans with increasing complexity.
Due to the complexity of such a system, current flight processor technology will not be sufficient to achieve the full set of autonomous functions onboard for robotics. Hardware limitations and software execution efficiency can be improved onboard through planned upgrades as relevant technology becomes available, but, for now, ground support must provide additional context information to allow autonomous re-planning.

Conclusions

Autonomy is essential to the success of future exploration missions, and autonomous systems come in many forms. Careful consideration of autonomous functions that need to be performed by a spacecraft must be completed early in the mission design in order to create and train a system that is capable of being operated in the conditions required. The operational mindset must be built into spacecraft that are going beyond low-Earth orbit.

The successful integration of autonomous capabilities into a spacecraft start with a solid approach to systems engineering, because the ability to operate independently from ground or crew requires knowledge and control of the entire spacecraft. An integrated system design is essential to the success and independence of the overall system.

Practically, this means that system designers should consider the following during spacecraft design. First, design choices that result in a reduction in system complexity will enable more robust reasoning about the system. Systems that are less complex need less complex algorithms to reason about them, and as a side benefit, will also allow more human visibility into system reactions. Along the same point, reductions in dependencies between subsystems enables more local, component-based decision making. This will also enable less complexity in autonomous solutions due to the inherent reduction in the overall state space and fault trees [4].

References

[1] Nancy G. Leveson, Role of Software in Spacecraft Accidents, Journal of Spacecraft and Rockets (2012) 4-6.

[2] What is the Software Systems domain? 1-August 2018, https://www.esa.int/Our_Activities/Space_Engineering_Technology/Software_Systems, (accessed 10.09.19).

[3] Kevin W. Lewis, Stephen Peters, Kurt Gonter2, Shaunna Morrison, Nicholas Schmerr, Ashwin R. Vasavada, Travis Gabriel, A surface gravity traverse on Mars indicates low bedrock density at Gale crater, Science Vol 363, Issue 6426 (2019) 2-3.

[4] Badger Julia, Higbee Don, Kennedy Tim, Vitalpur Sharada, Sargusingh Miriam, Shull Sarah, Othon Bill, Davies Francis J., Hurlbert Eric, Spacecraft

Dormancy Autonomy Analysis for a Crewed Martian Mission, Spacecraft Design, Testing and Performance (2018) 6, 28-29, 95-97, 109-115, 123, 125.

[5] MIL STD 1553 in military avionics. 20 December 2016, http://mil-avionics.com/benefits-mil-std-1553-provides-military-avionics/, (accessed 10.09.19)

[6] Donohoe, Gregory W. and Lyke, James C, Reconfigurable Computing for Space, Aerospace Technologies Advancements (2010) 14-15.