Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 16 (2013) 747 – 756

Procedia Computer Science

Conference on Systems Engineering Research (CSER'13)
Eds.: C.J.J. Paredis, C. Bishop, D. Bodner, Georgia Institute of Technology, Atlanta, GA, March 19-22, 2013.

The Evolution of Software and Its Impact on Complex System Design in Robotic Spacecraft Embedded Systems

Roy Butler[a]*, Michael Pennotti[b]

[a]*Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109, USA*
[b]*Stevens Institute of Technology, Castle Point on Hudson, Hoboken, NJ 07030, USA*

Abstract

The growth in computer hardware performance, coupled with reduced energy requirements, has led to a rapid expansion of the resources available to software systems, driving them towards greater logical abstraction, flexibility, and complexity. This shift in focus from compacting functionality into a limited field towards developing layered, multi-state architectures in a grand field has both driven and been driven by the history of embedded processor design in the robotic spacecraft industry.

The combinatorial growth of interprocess conditions is accompanied by benefits (concurrent development, situational autonomy, and evolution of goals) and drawbacks (late integration, non-deterministic interactions, and multifaceted anomalies) in achieving mission success, as illustrated by the case of the Mars Reconnaissance Orbiter. Approaches to optimizing the benefits while mitigating the drawbacks have taken the shape of the formalization of requirements, modular design practices, extensive system simulation, and spacecraft data trend analysis. The growth of hardware capability and software complexity can be expected to continue, with future directions including stackable commodity subsystems, computer-generated algorithms, runtime reconfigurable processors, and greater autonomy.

Keywords: Software, Complex Systems, Spacecraft

* Corresponding author. Tel.: +1-818-393-5844; fax: +1-818-354-5118.
E-mail address: roy.butler@jpl.nasa.gov.

1. Spacecraft Function and Resources

Space technology is a key contributor to the evolution of hardware and software design with a focus on the domains of system control, sensors, and telecommunications applied to the fields of robotics and embedded systems. The consistent growth of hardware performance in accordance with Moore's Law [1], coupled with the reduced computational energy requirements described by Koomey's Law [2], has fueled the expansion of software systems in terms of both size and complexity. Where programmers once had to focus on the utility of each digital bit, limiting the scope of their algorithms, they are now free to extend features utilizing higher level coding languages, more complex frameworks, and networked groups of systems, backed by the expectation that the next generation of hardware will provide increased resources.

Spacecraft system control, encoded in sequences of events, has evolved from hard-coded tables defined in hardware entirely ahead of launch, towards conditional event sequences in software, modifiable over the course of the mission should the need or new opportunity arise. Important events, including thruster burns and instrument activations, also once hard-coded and/or triggered from the ground, can now be software defined and incorporate greater condition-based state in determining activation due to the increase in computational power and memory provided by the hardware [3].

Sensor design, while still specialized in the aerospace industry, has evolved from simple timed analog captures towards complex on-board data acquisition management, digitization, data processing, and science return evaluation. An example from the past is the Soviet space program's Mars-3 spacecraft, which recorded images on a limited film supply according to timed sequences. Upon its planetary arrival in December 1971, which ill-fatedly coincided with heavy dust storm activity, a sizable portion of its science return was wasted imaging the nearly opaque Martian atmosphere [4]. With the move to rewritable digital storage systems and on-board observation quality analysis, modern platforms like the Mars Exploration Rovers can take, store, and prioritize continuous data over the course of a mission for return to Earth [5].

The wireless telecommunication field has rapidly advanced, from low-gain "bent pipe" one-shot repeater capability of transmissions to the modern digital high-gain, intelligent noise reduction, store-and-forward data transmission, including conditional bandwidth and retransmission features, all made possible through the greater computational abilities at the nodes and upgradable software-defined protocols. Such capabilities would have greatly assisted the Mars-3 mission, which also faced issues with its transmitter heating unsafely during extended use, necessitating it to send back its image data at lower fidelity resolution (255 vs. 1000 line mode) [6].

Counting the Software Lines of Code in a project is a useful, if imperfect, metric which illustrates the growth trend in complexity over time of the spacecraft systems we develop, see Fig 1 [7]. What began as single purpose, specialized hardware systems for these functions has grown to become complex software architectures running atop mostly commodity-based spacecraft embedded processors. This, as will be described, is not always a benefit as these now complex interacting subsystems increase the range of possible failures, as well.

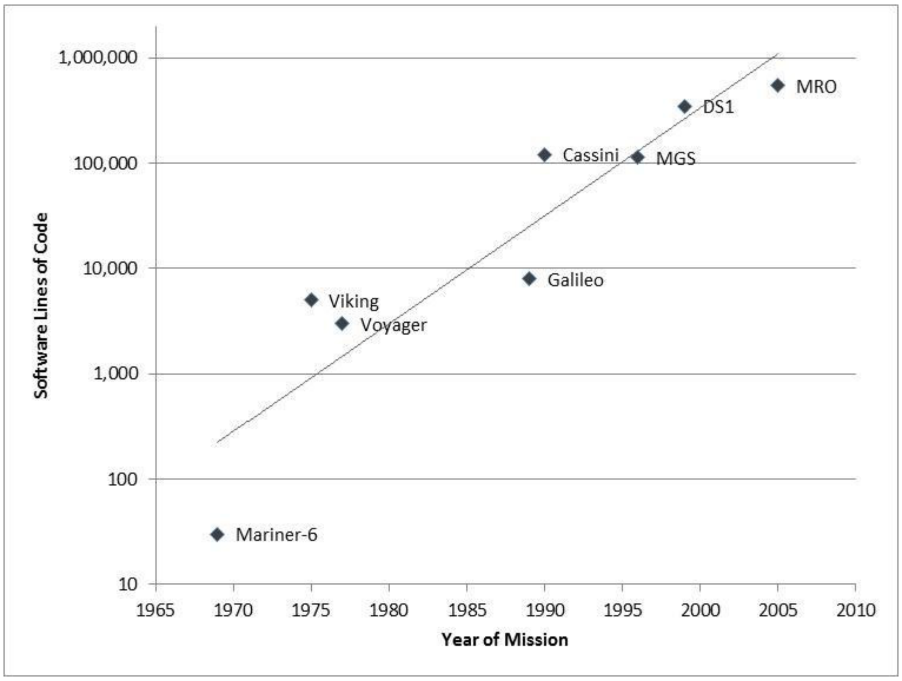

Fig. 1. Growth in Software Lines of Code

2. Trade-Offs in Systems Design

The combinatorial growth of interprocess conditions is accompanied by benefits (concurrent development, situational autonomy, and evolution of goals) and drawbacks (late integration, non-deterministic interactions, and multifaceted anomalies) in achieving mission success, as illustrated by the case of the Mars Reconnaissance Orbiter.

The Mars Reconnaissance Orbiter is a deep space satellite, launched in 2005, circling Mars with an array of science instruments, including cameras, spectrometers, and radar. Its design and operation was a joint effort of JPL and Lockheed Martin, under NASA contract, as the next step in a coordinated Mars program to determine the planet's geological past, including the inferred loss of surface water and the majority of its atmosphere. Data from the mission includes gravity modeling, surface mapping, subsurface stratigraphy, and weather monitoring. The satellite also serves as a telecommunications relay to landed assets, presently the MER "Opportunity" and recently-landed MSL "Curiosity" rovers [8].

2.1. Concurrent Development vs. Late Integration

The upfront definition of subsystem and instrument interfaces allows for the concurrent development of software and hardware by separate teams. Benefits of this approach include a shorter production cycle and therefore, potentially reduced staff costs. Drawbacks are the possibility of late integration accompanied by reduced verification, should rework cause delays to one or more of the instruments making it unavailable when originally estimated. There is a tendency to assume non-critical (but still serious) problems can be dealt with during integration or even post-launch with software updates, which while often true, results in a loss of the cost benefit and adds operational complexity through the introduction of hazard states through which the system cannot be operated until the situation is resolved.

Such events were experienced in the development of the Mars Reconnaissance Orbiter project, when FPGAs for several subsystems required replacement, delaying several of the instruments for integration. This led to a rushed schedule to meet its launch window for Mars, which if missed would have resulted in a possibly unfundable two year delay. Nominal electromagnetic interference testing was performed, but extensive characterization and testing of instrument interference with its Electra UHF relay radio was delayed until post-launch, given that the Software-Defined Radio (SDR) could receive software updates to incorporate any necessary band filters. This assumption has held true to a great extent and relay with Martian rovers has been an on-going success, but at the cost of a great amount of follow-on software development and test [9, 10].

2.2. Autonomy vs. Determinism

Building situational autonomy into spacecraft software, a sense of self-health, objective state awareness of relevant conditions, and flexible science operations timelines are among the benefits complex software can provide given a fixed set of hardware resources, all of which lead to intelligent spacecraft fault-tolerance and an increase in the amount and quality of science data return.

Rather than running exclusively to timed schedules, current spacecraft like the Mars Reconnaissance Orbiter use ephemeris data to provide them with navigational information (location, trajectory), from which they optimally recognize and initiate events. A command that would be proposed in a time-only domain with the form:

"Photograph at an 8 degree angle at 11:00"

can now be more accurately defined, producing better results in a computed evaluation incorporating both the time and spatial domains with the form:

"Photograph target X, centered, when closest on next orbit"

This flexibility and overall performance can be further increased by running different classes of code (i.e. instrument control, communications, and data management) in separate on-board software virtual machines, each restricted to their sandbox of task-relevant commands. While in reality these may all possibly execute on the same CPU, their non-interacting nature allows for a separation of concerns in planning between their respective ground team personnel. Many recent missions, including the Mars Reconnaissance Orbiter, implement this architecture using JPL's Virtual Machine Language abstraction [11].

Problems with an autonomous approach arise when assumptions about the relationship between multiple spacecraft operations and their context are inaccurate or misunderstood. Two observations by separate instruments may collide, physically or logically. For instance, a spacecraft roll angle to take a visual observation may interfere with the best target angle for UHF radio relay with one of the rovers. Checks for logical conflicts can be built into the ground-built sequence planning process, but these then need to evolve and be updated as operations change, as well as having reduced the on-board optimization. Issues have also arisen with shared spacecraft data storage space and downlink budgets, as the separate instrument operations teams mostly make their plans independently, which then get merged for on-board execution. Occasionally, the combination of heavy observation periods by multiple instruments coupled with a possible Deep Space Network (DSN) receiver antenna outage can result in the saturation of on-board science data storage, to which not all of the instruments were designed to handle graciously [12].

2.3. System Evolution vs. Stability

The ability of spacecraft system software to evolve via updates while in flight allows for new opportunities and mission goals to be achieved, as collected data leads to new insights into how to perform operations and also when new responsibilities are added to the mission (i.e. relay support of a new rover).

One example on the Mars Reconnaissance Orbiter is with its Compact Reconnaissance Imaging Spectrometer (CRISM) instrument, which was originally designed to determine planet surface element chemistry. Its standard operations have recently been extended to include atmospheric “limb scans” for tracking changes in Oxygen levels on the day and night sides of the planet across the seasons [13]. This has facilitated atmospheric composition and flow research, including comparison with simultaneous data sets taken with the spacecraft’s Mars Climate Sounder (MCS) instrument, which measures atmospheric humidity, dust, and temperature profiles using thermal imagery.

Finally, relay telecommunication for the Mars landers was designed to allow change over time. The Mars Exploration Rovers “Spirit” and “Opportunity” were already on the planet when Mars Reconnaissance Orbiter arrived, but since then the Phoenix lander and Mars Science Laboratory rover missions have reached the planet. These later missions have extended their radio protocols to include features like Adaptive Data Rate, where two communicating radios may autonomously adjust their bandwidth in the presence of clear or high error-rate channels and Auto Retransmit, where dropped data over UHF will be recognized and sent again without requiring ground intervention [14].

The drawback to allowing system evolution comes from the loss of stability in a predetermined baseline operations plan. New anomalies become harder to characterize and compare with the old, because all things have not remained equal – operating in different states and scenarios than before. This also incurs maintenance in keeping contingency plans up-to-date, as the methods in recovering from anomaly to a new baseline state change, so previous experience may not apply. This has been the case on Mars Reconnaissance Orbiter, necessitating changes to its safe mode recovery operations.

3. Dealing with Software Complexity

Approaches to optimizing the benefits while mitigating the drawbacks have taken the form of the formalization of requirements, modular design practices, extensive system simulation, and spacecraft data trend analysis.

3.1. Formalization of Requirements

Requirements design enumerates specifications derived from the mission needs, goals, and objectives - allocating at which level they need to be addressed in a top-down dependency tree. The rationale for each requirement, especially in the case of numeric quantities, is necessary to allow proper weighting in their respective trade space and to provide for background knowledge propagation across the teams. Requirements provide the logic behind what needs to travel between subsystem interfaces, as well as the complete framework within which decisions are made, before anything is built, about whether those interfaces are necessary and sufficient to the purpose of the overall mission.

Top-level requirements start with the system: what does it need to do. Then, each descending level defines what parts are necessary to fulfill the level above, branching into subsystems, and components, while leaving implementation specifics to the respective engineering teams. Every entry should be traceable to fulfilling a set of higher-level needs above it and creating the necessity of a set of needs below it at a more detailed level, see Fig 2. Changes made to a requirement then get traced up and down the chain to measure the effects and whether they are compatible with the remainder of the system [15].

Properly defined and managed requirements are the means by which a complex project is created, composed of numerous subsystems of which no single person or team can be an expert. It is a reference table, on which mission scope and subsystem design trades can take place with visibility into how the part affects the whole, allowing a reasoned approach to work its way throughout a system too large to contemplate at once.

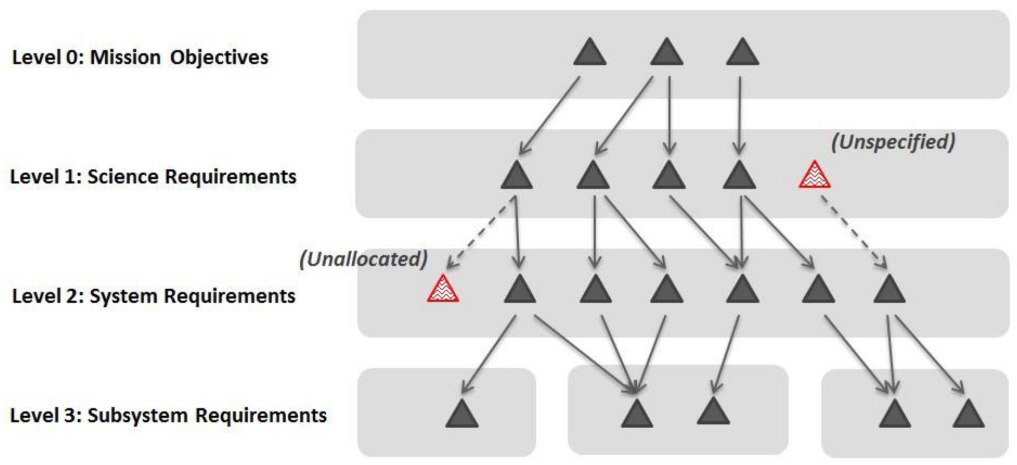
Level 0: Mission Objectives
Level 1: Science Requirements
Level 2: System Requirements
Level 3: Subsystem Requirements
(Unspecified)
(Unallocated)

Fig. 2. Requirements Traceability

3.2. Modular Design Practices

Modular design practices in both hardware and software, through high internal cohesion and loose external coupling of component design, reduce dependencies and clarify the state space, aiding in the development and integration of the overall system. They also allow for the use of swappable, lower functionality fail-safe components, to be activated in the event of significant on-board errors in order to maintain baseline spacecraft power, thermal, and communication status while the anomaly is investigated and resolved through ground staff personnel intervention.

Two useful methods for breaking the complete system down into modules of high cohesion and loose coupling are to define the system in terms of sequential binding or functional binding. Sequential binding can be achieved by flowcharting the events performed by the spacecraft (i.e. take observation, transmit data to Earth), then defining modules around the logical flow blocks of the diagram. A functional binding representation can be obtained from a data-centric point of view: "Where is this piece of data used?" Those data commonly used in conjunction (i.e. solar array parameters, battery state) form groupings around which modules can be designed. The sequential binding and functional binding methods can be used in combination and they apply to both hardware and software [16].

Once modules are created using workflow or data-centric abstractions, complexity can be stemmed in the software realm by restricting inter-module calls to the form of message passing. That is, when one module requests a service from another, it does so in the form of a message, which the second module can evaluate whether or not to comply with based on tracked state and a range check of parameters (design-by-contract). This differs in kind from direct function calls between modules and allows the framework for containing and recognizing errors within a portion of the system and handling the situation accordingly, rather than allowing a rapid spread of faulty state throughout the entire software collection [17].

Spacecraft safety and health are also served by modular design, through the implementation of redundant modules to perform critical activities: one or more standard modules for nominal mode activities and alternate, minimal functionality modules to protect key systems while relying on the least on-board state as possible, for off-nominal mode spacecraft preservation. Having redundant standard modules, especially in the case of hardware, allows for failure of components over the course of the mission, since these are mostly non-serviceable in the space environment [18]. The existence of swappable modules varying in degree of functionality often continues to take shape in software post-launch, including the refinement of the nominal and minimal cases, as well as for the development of more complex technology demonstration software deployments, once the mission's primary objectives have been met.

3.3. System Simulation

Simulation evolves through the project development phases starting with mission design software (i.e. SOAP, STK) for early phase feasibility studies, mid-phase engineering design and test software (i.e. CAD, LabVIEW), evolving into an operations phase complete spacecraft system hardware-in-the-loop testbed with flight computer and instrument engineering models. As verification of hardware interfaces and software performance is completed, validation of the overall system to meet the mission design requirements takes place, prior to launch. Then, over the course of the mission, beginning with launch itself, spacecraft command sequences of important events are run through the testbed ahead of time to validate operation plans, software interactions, and hardware timing. The testbed can also be used in the proactive generation of contingency plans, to test recovery mechanisms following intentional software-injected (or even hardware-injected) failures [19].

Creating an individual spacecraft subsystem, whether it is a physical sensor instrument, sequence control software, or the underlying power system is a bit of a dilemma due to the lack of pre-existing interfacing subsystems, given separated teams and parallel design schedules. Herein lays the importance of specifying hardware and software interface agreements ahead of implementation, so that each team can build their portion around a low to moderate fidelity simulator of all other interacting portions. Assuring a shared understanding of "what's on the other side" is crucial early on, as differences and problems grow harder and more costly to work out further into design, as components have solidified and been built on top of internal dependencies.

As individual teams' work on the development of their engineering model prototype completes, the units and spacecraft testbed system and bus are integrated, yielding the initial hardware-in-the-loop simulator for nominal sequences to be executed. Any incompatibilities are characterized and reworked in the interfacing subsystems. Development and integration of flight units increases at this phase as the spacecraft itself is built around the testbed model, with the addition of flight-only components (i.e. complete solar panels, fuel system and thrusters).

From launch and onwards, the ground testbeds are used to validate nominal sequences, software updates, and resolve anomalies. This reduces flight risk and provides a test environment for system evolution to incorporate new features and achieve emerging mission objectives. It also allows the engineering team to compare differing approaches to anomaly resolution in a safe environment, in order to determine ground control's best course of action.

3.4. Data Trend Analysis

Tracking and analysis of trends in spacecraft engineering data provides insight into underlying processes and can aid in the discovery of hidden states, those unplanned for in the original system design process, which reveal themselves through later correlations among on-board events. General spacecraft housekeeping data including timing, resource utilization, power, and thermal fluctuations should all be reviewed and retained to form a baseline of the expected spacecraft performance envelope. Then, as anomalies arise over the course of the mission, the surrounding data leading up to that point in time can be compared against past records to pick out the differences in the search for probable cause. While this is straight-forward to perform for a handful of variables, it quickly becomes overwhelming in scope when comparing values from multiple subsystems for extended periods of time. It's often the case that not a single instantaneous value elicits a failure, but rather the dynamics in the direction and rates of change among variables giving rise to the unexpected.

Statistical software packages like R [20] and its PerformanceAnalytics module bring much needed clarity to the task. An example is the correlation matrix chart, which plots histogram and kernel density estimations for each variable along the diagonal, scatterplots per pair of variables beneath, and statistical absolute correlation values and significance estimates above. Visually picking out deviations from the norm, including spotting multiple contributors, becomes tractable by this approach, rather than pouring through streams of textual raw telemetry data, see Table 1 and Fig 3.

Trends discovered by these means, which form tell-tale signs of pending faults, can then be compiled into an automated ground data system monitoring and alert rule set. This will perform the rote work of evaluating and logging flight engineering data on a minute-by-minute basis, paging operations staff in the event parameters extend outside the norm.

Table 1. Telemetry Set

time	angle	obs	temp	sun	radio	power
0	0	0	200	1	0	40
5	-12	1	220	1	1	50
10	-6	1	240	1	1	60
15	10	1	260	1	1	70
20	-5	1	280	1	1	80
25	7	1	280	1	1	90
30	12	0	280	1	1	100
35	0	1	280	1	1	100
40	3	1	280	1	1	100
45	7	1	280	1	1	100
50	-7	1	280	1	1	100
55	-4	1	280	1	1	100
60	0	0	280	1	0	100
65	0	0	270	0	0	95
70	0	0	260	0	0	90
75	0	0	250	0	0	85
80	0	0	240	0	0	80
85	0	0	230	0	0	75
90	0	0	220	0	0	70
95	0	0	210	0	0	65
100	0	0	200	0	0	60
105	0	0	200	0	0	55
110	0	0	200	0	0	50
115	0	0	200	0	0	45
120	0	0	200	0	0	40

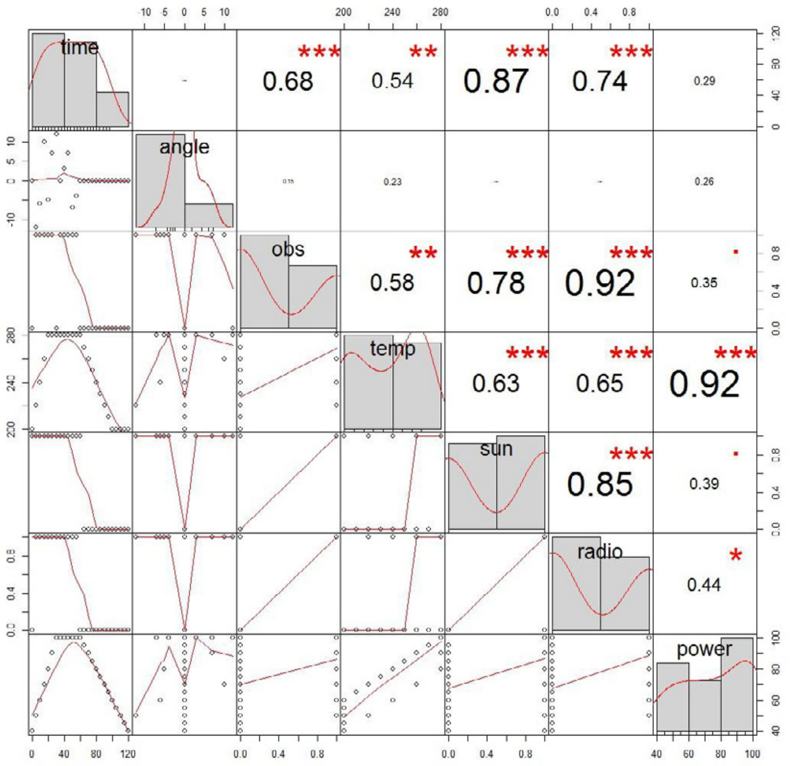

Fig. 3. Correlation Matrix Chart

4. Future Directions

The continued growth of hardware capability and software complexity can be expected, with future directions including stackable commodity subsystems, computer-generated algorithms, runtime reconfigurable processors, and greater autonomy.

Standardization of spacecraft subsystems and interfaces is a goal which is already taking place for Earth-orbiting satellites, that number in the thousands. Deep space missions can leverage some of these, but given their unique thermal, radiation, and signal delay environments, one-off solutions continue to take precedence. NASA's X2000 Program was one initiative working towards this goal [21].

The next stages of flexibility and optimization may be to turn software algorithms onto the hard problem of algorithm design themselves through genetic programming. This field has already seen some exploration in the development of link budget and image compression handlers [22]. Another phase to this self-modifiability could be on-board development of FPGA circuit bitfiles, introducing the possibility of runtime reconfigurable processors to most efficiently execute the operations at hand [23].

Fully autonomous spacecraft with active on-board intelligence and the ability to determine science targets on their own based on the situation observed (i.e. storms, fires) is next step flight software developers are working towards. Early technology demonstrations have already taken place, such as on-board NASA's EO-1 spacecraft with the Autonomous Science Agent software, after having completed its primary mission [24].

5. Summary

Systems are becoming increasingly more complex, both on Earth and in the robotic spacecraft embedded systems we send to distant planets. The ability to create such multifaceted systems, with their inherent ability to evolve, has come from the growth in computer hardware performance and reduced energy requirements, allowing software functionality to expand in flexibility and scope. This explosion in growth has benefits and drawbacks, which have been illustrated through examples of trade-offs during development, operation, and handling change. Relying on good requirements, modular design, system simulation, and trend analysis are classical systems engineering approaches – all of which shed insight and apply coherence to the comprehensive state, managing the increased complexity caused by the proliferation of software. Commodity subsystems and self-modifiable software are possible next steps towards greater autonomy and more scientific return from robotic spacecraft.

Acknowledgements

This paper was submitted in partial fulfillment of a Master's of Engineering in Systems Engineering at Stevens Institute of Technology, Hoboken, NJ USA.

References

1. R. Rumelt, O. Costa, “Gordon Moore’s Law (POL-2003-03),” Anderson School at UCLA, 2003.
2. J.G. Koomey, S. Berard, M. Sanchez, H. Wong, “Implications of Historical Trends in the Electrical Efficiency of Computing,” IEEE Annals of the History of Computing, Vol. 33, Issue 3, March 2011.
3. P. Stakem, The History of Spacecraft Computers from the V-2 to the Space Station, PRB Publishing, 2011.
4. P. Ulivi, D.M. Harland, Robotic Exploration of the Solar System, Part 1: The Golden Age 1957-1982, Springer, 2007.
5. M.W. Maimone, P.C. Leger, J.J. Biesiadecki, “Overview of the Mars Exploration Rovers,” IEEE Space Robotics Workshop, April 2007.
6. StrykFoto Mars-3 web site, http://www.strykfoto.org/mars3.htm
7. D. Dvorak, “NASA Study: Flight Software Complexity,” Workshop on Spacecraft Flight Software (FSW-08), November 2008.
8. M.D. Johnston, J.E. Graf, R.W. Zurek, H.J. Eisen, B. Jai, “The Mars Reconnaissance Orbiter Mission,” IEEE Aerospace Conference Proceedings, 2004.
9. T.J. Bayer, “Mars Reconnaissance Orbiter In-Flight Anomalies and Lessons Learned,” IEEEAC Paper #1451, 2007.
10. T.J. Bayer, “Mars Reconnaissance Orbiter In-Flight Anomalies and Lessons Learned: An Update,” IEEEAC Paper #1086, 2009.
11. C.A. Grasso, “The Fully Programmable Spacecraft: Procedural Sequencing for JPL Deep Space Missions Using VML (Virtual Machine Language),” IEEE Aerospace Applications Conference Proceedings, March 2002.
12. R. Gladden, F. Fisher, T. Khanampornpan, B. Waggoner, R. Thomas, D. Wenkert, “NIPCs: The Operational Sandbox of Science Commanding,” AIAA Paper #2006-5739, 2006.
13. R.T. Clancy, M. Wolff, M. Smith, T. McConnochie, F. Lefevre, S. Murchie, “CRISM limb observations of Mars dayside O2 singlet delta emission during 2010-2011,” European Planetary Science Congress (EPSC), 2011.
14. C.D. Edwards, T.C. Jedrey, E. Schwartzbaum, A.S. Devereaux, “The Electra Proximity Link Payload for Mars Relay Telecommunications and Navigation,” 54th International Astronautical Congress, 2003.
15. I. Hooks, K. Farry, Customer-Centered Products, Creating Successful Products through Smart Requirements Management, AMA, 2001.
16. W. Stevens, G. Myers, L. Constantine, “Structured Design,” IBM Systems Journal, Vol. 13, No. 2, 1974.
17. Jet Propulsion Laboratory, “JPL Institutional Coding Standard for the C Programming Language (JPL DOCID D-60411),” March 2009.
18. E.H. Seale, “The Evolution of a SPIDER,” Proceedings of the 2003 IEEE Aerospace Conference, March 2003.
19. J. Eickhoff, Simulating Spacecraft Systems, Springer, 2009.
20. D.B. Wright, K. London, Modern Regression Techniques Using R: A Practical Guide for Students and Researchers, SAGE, 2009.
21. L.J. Deutsch, C. Salvo, D. Woerner, “NASA’s X2000 Program,” Acta Astronautica, Vol. 46, Issues 2-6, 2000.
22. J. Miller, P. Thomson, “Cartesian Genetic Programming,” Proceedings of the EuroGP2000, Springer-Verlag, 2000.
23. D. Meyer, “Runtime Reconfigurable Processors,” Chaos Communication Camp presentation, 2011.
24. S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castano, et al., “The EO-1 Autonomous Science Agent,” Proceedings of the 2004 Autonomous Agents and MultiAgent Systems Conference, 2004.