An Incremental Strategy for Spacecraft Flight Software Reuse

Gary M. Heiligman, T. Adrian Hill, Robyn L. LeGrys, and Stephen P. Williams
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road, Laurel, MD 20723-6099
mailto:Gary.Heiligman@jhuapl.edu

Abstract

This paper describes a process of spacecraft flight software development that reuses requirements, designs, and implementations. Beginning in 1997, The Johns Hopkins University Applied Physics Laboratory (JHU/APL) developed the flight software for five different space missions: (1) Thermosphere, Ionosphere, Mesosphere Energetics and Dynamics (TIMED), (2) COmet Nucleus TOUR (CONTOUR), (3) MErcury Surface, Space ENvironment, GEochemistry, and Ranging (MESSENGER), (4) Solar TErrestrial RElations Observatory (STEREO), and (5) New Horizons, a mission to Pluto and beyond. JHU/APL met strict constraints of budget and schedule by reusing products from each mission for the following one in a successively more comprehensive fashion. Keys to the success of this reuse are consistent external interface protocols, rigorous requirements management, retention of the original development documentation, use of a consistent development process, and a group organization that fosters reuse. The reused packages include bootstrap, task scheduling, uplink, command handling, autonomy, and 1553 bus support.

1. Introduction

The Johns Hopkins University Applied Physics Laboratory (JHU/APL) has been developing spacecraft flight software ever since the first general-purpose computers were flown on spacecraft [1]. This software is of three types.

(1) Command and data handling (C&DH) programs process telecommands, transmit telemetry, and provide onboard data storage.
(2) Guidance and control (G&C) programs determine the spacecraft's attitude and trajectory and control them.
(3) Instrument programs control and take data from the payload and usually run on a dedicated instrument data processing unit (IDPU).

IDPU software is delivered with the instruments; C&DH and G&C software are provided with the spacecraft bus.

In 1997, JHU/APL completed the critical design review (CDR) for the *TIMED* mission. Mission-level CDRs for three other flight projects followed in quick succession: *CONTOUR* in 2000, *MESSENGER* in 2002, and *STEREO* in 2003. The CDR for a fourth mission, *New Horizons,* will also be held in 2003. Each of these missions requires C&DH and G&C software; some require instrument software as well.

As one of its unique capabilities, JHU/APL's Space Department provides experience and expertise in developing and testing spacecraft flight software. A group of less than 30 developers delivers the software for these and future missions. Software reuse plays an important role in the way the group creates these computer programs.

At JHU/APL, reuse consists of taking the requirements, designs, source code, and associated documentation from past missions as the starting point for each new mission. The developers of *CONTOUR* and later missions have succeeded in reusing incrementally greater portions of the software from past missions in a process that we describe here.

2. Motivations for reuse

2.1. Reliability

Because failure of a critical computer program could easily lead to the loss of the entire mission, reliability is paramount for spacecraft flight software. JHU/APL therefore takes a conservative approach to the development of both hardware and software. For example, JHU/APL uses only space-qualified, radiation-hard processors and memory in its spacecraft. The development tools (compiler, locator, and operating system) must also be highly reliable. "Flight heritage"—the proven capability of an article to perform to specifications in the harsh environment of space—is considered an asset in both hardware and software.

In software, flight heritage is obvious when source code is reused from previous missions. However, flight heritage is also valuable for designs and requirements, because correctness of these products is just as critical to mission success as bug-free software.

2.2. *Volume*

The C&DH and G&C programs for each mission require tens of thousands of lines of code. Starting in 1997, JHU/APL has had to develop spacecraft flight software at a rate of approximately one new mission per year. Developing this much software anew for each mission would strain JHU/APL's in-house capability even if time were not a factor.

2.3. *Time*

Deep-space missions must be launched on a schedule dictated by the dynamics of the Solar System. In order to be ready for launch, the software must be developed, integrated, and tested. Thus reuse often becomes essential even if the developers would like to make changes (for efficiency, maintainability, reusability, etc.), because there is simply not enough time.

3. Barriers to Reuse

3.1. *Differences in mission-level requirements dictate different solutions*

The most important barrier to reuse is the different requirements at the mission level that lead to different software architectures. The five missions discussed in this paper are the following.

(1) *TIMED* is the first mission in NASA's Solar Terrestrial Probes program. It studies the Earth's upper atmosphere with four remote-sensing instruments. It has performed its scientific mission successfully since its launch on December 7, 2001.

(2) *CONTOUR* was a Discovery-class mission to explore the diversity of comets. It performed flawlessly for six weeks of phasing orbits, but it was destroyed on August 15, 2002, during the solid-rocket motor burn that was to place it in heliocentric orbit.

(3) *MESSENGER* is a Discovery-class mission to fly by and orbit Mercury. It is manifested for launch on March 10, 2004.

(4) *STEREO* is the third mission in NASA's Solar Terrestrial Probes program. It consists of two identical spacecraft in different heliocentric orbits. Launch (from a single vehicle) will occur in November 2005.

(5) *New Horizons* is the first mission to Pluto-Charon and the Kuiper Belt of rocky, icy objects beyond. Launch is planned for January 2006.

We summarize the differences for these five missions in the Appendix. The number and types of processors, how they are interconnected, and the lifetime of the mission all affect the software in fundamental ways. The different mission drivers, though, are often the most critical barriers to reuse, because they may dictate that a solution that worked for past missions is untenable for the new mission. They may affect the level of autonomy, the timing constraints, or the basic assumptions under which the software must function.

3.2. *Focus on individual missions means limited effort for reuse per se*

Flight software development at JHU/APL is funded almost entirely by the individual missions. Mission managers are understandably reluctant to devote precious dollars allocated to their own mission to fund reusability improvements that primarily benefit future missions. And missions not yet funded for implementation generally are unable to provide the resources to ensure that a flight software product is developed for reusability. Thus there is no ready pool of money for reuse *per se*. The staff organization within JHU/APL takes on the responsibility of fostering reuse as a part of its primary goal of developing mission-specific software.

4. Framework for reuse

4.1. *Hardware architecture*

The spacecraft before *TIMED* distributed the data processing over hardware in several different ways and used a variety of processors and implementation languages. Since then, however, three features have contributed to the stabilization of G&C and C&DH software.

4.1.1. The integrated electronics module (IEM). On *TIMED* and all subsequent missions, a PCI backplane connects the command and data handling (C&DH) processor, solid-state recorder, and transponder interfaces in one physical box: the IEM. Consistency of uplink protocols is in part a result of this decision. Most missions have two IEMs for full redundancy.

4.1.2. The UTMC SμMMIT chip. *TIMED* and all subsequent missions have used a consistent protocol and hardware set to communicate between the processor and many system components: the MIL-STD-1553B Notice 2 bus and the SμMMIT integrated circuit family from UTMC. Consistency of this interface enables much of the external communication software to be reused.

4.1.3. The power distribution unit interface. On *CONTOUR* and all subsequent missions, electronics in the power distribution unit provide the interface to the power system electronics and some attitude control hardware (e.g., thrusters, sun-angle sensors, reaction wheels, and solar-array drives). It translates 1553 bus messages into command signals to the hardware and translates data signals into 1553 bus messages. Consistency of this interface also leads to software reuse.

4.2. *Development methodology and language*

A shared methodology and language are essential for reuse. The flight software developers use a "waterfall" development methodology, as shown in Figure 1.

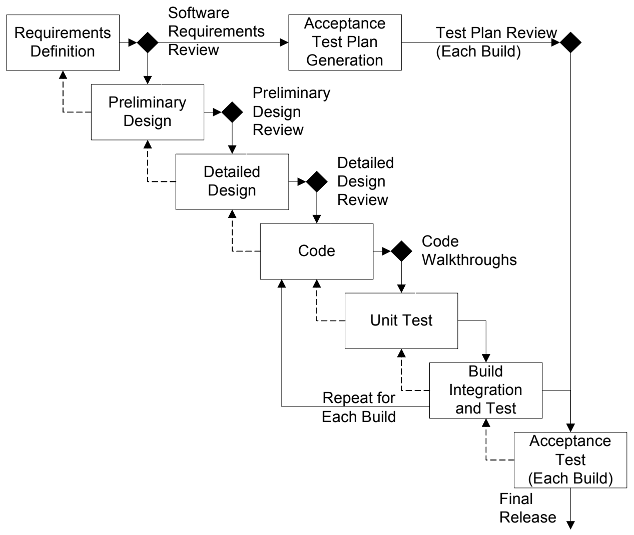
Requirements Definition
Software Requirements Review
Acceptance Test Plan Generation
Test Plan Review (Each Build)
Preliminary Design
Preliminary Design Review
Detailed Design
Detailed Design Review
Code
Code Walkthroughs
Unit Test
Build Integration and Test
Repeat for Each Build
Acceptance Test (Each Build)
Final Release

Figure 1. The flight software development methodology

The design methodology, based on the process of Shumate and Keller [2], is tailored for the development of mission-critical, embedded, real-time software. Program definition language (PDL) is used in addition to the Shumate-Keller graphical notation to describe functional designs. The designs are implemented in the C programming language as set forth in the ANSI 1990 standard [3].

4.3. *Reviews and code walkthroughs*

One of the most important contributing factors to effective reuse at JHU/APL is the participation of developers from other missions on each mission's review panels. The Space Department's policies and procedures dictate that for each mission all requirements, all designs, and all new code must be peer-reviewed. The reviews follow a set procedure, and every review panel must include a participant not involved with the development of that mission.

Thus developers of later missions learn about products from earlier missions. Often, this knowledge comes at a stage such that the product can be reused in the later mission with minimal effort.

When the outside reviewer is from a mission at a later stage of development, the reviewer may recommend that the developers reuse part or all of a product from an earlier mission.

Also, during the review of a partially reused design, the reviewers may find defects in a reused portion of the design. The corrective actions may apply to both the earlier and later missions; thus, corrective actions themselves can be reused.

4.4. *Process documentation*

Each mission requires a software development management plan (SDMP) tailored to its unique needs. Large portions of this document are often reusable.

Perhaps more importantly, consistency of the software *modus operandi* means that the developers need not learn a new way of doing business when they begin work on a mission. It is particularly important that the graphical design notation not change, because these products are both time-consuming to produce and nontrivial to review.

4.5. *Development tools*

For a mission to reuse the development products of a preceding mission efficiently, paper copies are inadequate. The reason is that the products inevitably require tailoring for each new mission. JHU/APL addresses this problem in two ways: by using a consistent tool set across missions and by retaining the development products from past missions in their original form.

Table 1 is a list of the tools most heavily used for software development

Table 1. Development tools

Product	Tool	Vendor
Itemized requirements	DOORS	Telelogic
Diagrams	VISIO	Microsoft
Commands and telemetry	Excel	Microsoft
1553 bus schedules	Excel	Microsoft
Process documentation	Word	Microsoft
Presentation graphics	PowerPoint	Microsoft

In addition, where the hardware permits, JHU/APL also uses a consistent development environment. For example, *MESSENGER* and *STEREO* both use *VxWorks* 5.3.1 and *Tornado* 1.0.1 from Wind River Systems with the *Multi* 1.8.8 C compiler from Green Hills Software. *TIMED*, *CONTOUR*, and *New Horizons* use *Nucleus+* from Accelerated Technologies and the *Tasking* C compiler.

4.5.1. Retention of editable electronic copies. The documentation for each mission is delivered into a formal configuration control system. However, this system usually accepts documents in Microsoft Word or Adobe PDF format. For reuse it is critical that the products from each phase of the development project be retained *in an editable form* so that they can be tailored for the next mission. Thus the DOORS, VISIO, PowerPoint, etc. documents in their native formats are kept intact past the end of the mission so they can be reused.

4.5.2. Isolating changes to reused software. One of the biggest challenges to partial reuse is determining whether a change from heritage introduces a defect. At JHU/APL, the solution is to review the reused products (and the new ones) for each mission, with two exceptions. First, fully reused source code is not inspected. Second, the reused portion of partially reused source code is not inspected unless the interface or environment changes.

4.6. Group organization

The Embedded Applications Group of JHU/APL's Space Department is responsible for developing spacecraft flight software. This organization provides the physical space, desktop computers, and office software for the developers (the missions provide development host computers, development tools, and the target platforms). It also sets policies and procedures, provides training, manages the staff, and defines the culture in which the developers work. The individual developers are usually assigned to only one mission at a time, so it is through the group organization that reuse is fostered. Two factors assist in fostering reuse.

4.6.1. A culture of mutual assistance. If a project fails, JHU/APL views it as a failure of the entire organization. Thus developers are not motivated to segregate themselves on their own projects and responsibilities; instead, they are encouraged to communicate with developers on other projects and with subsystems outside of flight software on their own missions. The group management facilitates this communication by keeping the group apprised of the progress of each mission and encouraging the sharing of lessons learned. Developers can spend a limited amount of time attending meetings or answering questions not directly related to their primary mission without charging a special budget. This encourages a culture of mutual assistance, which is essential for reuse to succeed.

4.6.2. An emphasis on heritage as an asset. JHU/APL advocates that developers be able to answer the question "What did past missions do?" before creating their own new requirements, designs, or code. Because the design documentation from the previous missions is readily available, answering this question is typically easier than creating a new document.

5. Incremental reuse at each development phase

5.1. Reusable development products

Products of the development process are listed in Table 2. Those that are reused in whole or in part are in boldface.

Table 2. Reusable development products

Phase	Products
Requirements definition	**Itemized requirements** Acceptance test plan Context diagrams
Preliminary design	Task communication graphs Software architecture diagrams **Data flow diagrams** **Commands and telemetry** **Bus schedules**
Detailed design	**Task flow diagrams and PDL** **Method flow diagrams and PDL** **Package diagrams** Dependency diagrams **Header files**
Coding	**Source code files**
Unit test	**Unit test harnesses**
(All phases)	**Software development management plan** **Coding standards**

In general, the product types that are reusable are "building blocks"; those that are not reusable describe how the building blocks are put together. For example, the task communication graphs describe how all the flight software functions work together: because the spacecraft have different functional roles these diagrams are generally not reusable. On the other hand, task flowcharts describe the operation within a given functional area: these are often reusable because the functional task and its requirements are reused from mission to mission.

5.2. Requirements definition

An essential feature of JHU/APL's development process is careful management of requirements. Developers of each mission review software requirements to ensure that they are itemized, complete, consistent, unambiguous, verifiable, and traceable to the parent system specification. The requirements management tool (DOORS) organizes the requirements hierarchically, so developers of successive missions can select individual requirements, modules, sections, or even whole documents for reuse.

This type of reuse is enormously valuable because all the succeeding development phases can take as their starting point the products from a previous mission. Measures of requirements reuse are probably a better indicator of software heritage and reliability than metrics at later phases.

5.3. Preliminary design

The developers perform functional and structural decomposition during this phase, i.e., they identify tasks (threads of program execution) and packages (mutually dependent groups of functions and data structures). Many individual tasks are reused from mission to mission. But the task communication graph tends not to be very reusable because the way that the tasks interact with one another varies widely from mission to mission. Similarly, many individual packages are reused repeatedly. But the software architecture diagram that describes how packages depend on one another varies considerably for each mission.

Within packages and tasks, data flow diagrams and the data dictionaries are heavily reused. Reusing command and telemetry descriptions is valuable at this stage because the detailed designs and implementations follow directly from the description of the command.

5.4. Detailed design

The developers define the package diagram—the functions and data stores implemented by each package—during detailed design. This product is highly reusable from mission to mission, although functions are frequently added or removed during the development process. Application program interfaces (APIs) are also defined during this phase. When an API can be reused from a previous mission, all the detailed designs and code that use that API are more reusable. As with the software architecture diagram, the dependency diagram—the "who calls whom" graph—tends to be less reusable because the way that packages fit together varies from mission to mission.

In detailed design, developers represent the functional flow of tasks and methods with either graphical notation or pseudocode. These products are reusable either in whole or in part when the functional areas they implement are similar.

5.5. Code

Source code reuse is the most rudimentary form of reuse. Mission management often measures reuse at this level because it is a convenient metric for flight heritage. However, it is an inconsistent indicator of software reliability unless the products from preceding phases are reused. In other words, if heritage source code is used in new ways, new defects may manifest themselves.

Rather than dictate full reuse in such cases, JHU/APL advocates incremental reuse when the interfaces must be different but the underlying algorithms are the same. An example of this incremental reuse occurs when a resource is used by only one task in one design but shared by many tasks in a later design. When this pattern occurs the shared resource requires a protection mechanism (e.g., a mutual exclusion semaphore). This is added during package design, and the addition propagates through package diagrams, flowcharts, header files, and source code.

6. Reused functional areas

Table 3 provides a summary of the functional areas that have been or are being reused by the sequence of missions starting with *TIMED*. Items in italics achieved a level of reuse of 50% or more.

Table 3. New and reused functional areas

Functional area	New development(s)	Reuse
Bootstrap	TIMED	*CONTOUR* New Horizons
Bootstrap	MESSENGER	STEREO
Uplink	TIMED	*CONTOUR* *MESSENGER* *STEREO* *New Horizons*
Command executive and macros	TIMED	*CONTOUR* *MESSENGER* *STEREO* *New Horizons*
Task scheduler Preliminary design, detailed design, and code	CONTOUR	MESSENGER *STEREO* *New Horizons*
1553 bus support Requirements, preliminary design, and code	CONTOUR	*MESSENGER* *STEREO* *New Horizons*
Hardware-specific utilities Detailed design and code	TIMED	*CONTOUR* *New Horizons*
Hardware-specific utilities Detailed design and code	MESSENGER	*STEREO*
Autonomy and time-tagged rules	TIMED	*CONTOUR* *MESSENGER* *STEREO* *New Horizons*
Autonomy reverse polish notation	MESSENGER	*STEREO* *New Horizons*

6.1. Bootstrap

The bootstrap program takes the processor from its initial—possibly indeterminate—state after reset to the point that the spacecraft flight application can start.

On spacecraft that have fully redundant processors, the bootstrap program also provides access to the backup processor while the primary processor runs the flight application. The nature and extent of this capability is determined during requirements definition and affects the itemized requirements and preliminary design products.

Much of the detailed design is dependent on the processor and operating system, so there are two lineages in the later development phases. The *MESSENGER-STEREO* lineage uses the BAE RAD6000 processor and *VxWorks*; the *TIMED-CONTOUR-New Horizons* lineage uses the Synova Mongoose V processor and *Nucleus*+.

6.2. Telecommand packet handling

For spacecraft uplink, *TIMED* developed the interface to the spacecraft transponder and identified formats and protocols from the Consultative Committee on Space Data Standards (CCSDS). All missions following have held largely to these decisions, so there is a great deal of reuse of these products.

6.3. Command executive, macros, autonomy, and time-tagged rules

The developers of *TIMED* created requirements, designs, and code that defined how the spacecraft executes individual commands, command sequences (“macros”), and commands that must be executed at a specific time. This functional area has remained stable throughout the succeeding sequence of spacecraft. A key part of the *TIMED* design that is carried over to following missions is the API: the function calls and signatures needed to fit a new command into the design framework are the same.

6.4. Task scheduler

How tasks are scheduled and synchronized is one of the essential elements of preliminary design. The design developed for *TIMED* is used on succeeding missions, albeit tailored for the two different target environments. The source code must be modified for each mission to handle the differences in timing requirements and task lists.

6.5. MIL-STD-1553B bus support

Although the MIL-STD-1553 bus architectures and schedules are very different for each mission, the requirements for bus support are fairly similar; these and some preliminary designs are reusable. *TIMED* developers created a spreadsheet template to develop the bus schedules that following missions have reused.

At the detailed design level, however, each mission is unique. This is because each message type requires its own processing. Thus much of the detailed design and the application levels of code are not reusable.

Bus support software beneath this layer is more reusable. The function calls that exchange data with and control the SμMMIT chip are reused. There are actually two functional areas that are reused: one for the bus controller and another for the remote terminal.

6.6. Hardware-specific utilities

Detailed designs and code that provide a layer of abstraction to the hardware can be reused when the target hardware and O/S are the same from mission to mission. The bootstrap program and the task scheduler overlap with this functional area of reuse. Among the utilities that are reused are the following.

6.6.1. Non-volatile memory programming. *STEREO* reuses *MESSENGER*’s designs and code for writing to electrically erasable programmable read-only memory (EEPROM). *TIMED*, *CONTOUR*, and *New Horizons* use Flash memory for long-term storage, and there is extensive reuse of these designs and code as well.

6.6.2. Peak power tracking. The power distribution unit transmits data to the processor, which then calculates a voltage regulator setting that optimizes the power from the solar array and the charging of the battery. This algorithm and much of the software that implements it are reused from *TIMED* for *MESSENGER* and *STEREO*. *CONTOUR* used fixed power settings, and *New Horizons* uses a radioisotope thermoelectric generator (RTG), so they do not reuse this software.

7. Example of incremental reuse: autonomy

One criticism of “institutionalized” reuse is that it often stifles the ability to insert new technologies into already existing systems. JHU/APL has successfully applied the approach of incremental reuse in the area of flight software that performs autonomy by expanding the capability of the on-board software from mission-to-mission while maintaining significant reuse at both the design and coding levels.

The JHU/APL missions use an on-board autonomy engine that is capable of evaluating autonomy rules used for spacecraft health and safety. An autonomy rule contains a premise (e.g., battery temperature > 30°C), persistence limits (how long the condition must persist), and a response (usually a command or macro that is executed to respond to the condition). The autonomy rules are not compiled into the flight software; rather, they are designed by the spacecraft Safing Engineer and loaded to the spacecraft. The on-board autonomy engine flight software executes and acts on the loaded rules.

At the heart of the on-board autonomy engine is the Data Collection Buffer (DCB). The DCB is a collection of data points from all subsystems on the spacecraft that is populated by the flight software and is refreshed at a 1 Hz rate. DCB data include hardware data such as raw voltages, currents, and temperatures, and software data such as counters and flags. These points are all available for use by the Safing Engineer for autonomy rules premises.

7.1. *TIMED capability*

On *TIMED*, C&DH developers introduced the autonomy engine to the flight software. The software implementation provided support for performing simple arithmetic and relational operations of data points in the DCB. There were a limited number of predefined expression formats that could be used as the premise of the rule (e.g., "A > B AND C > D"). The software API included methods to evaluate the rule premises, increment persistence counters, fire commands and macros, as well as support enabling and disabling autonomy rules and resetting running persistence counters.

7.2. *CONTOUR enhancements*

Both developers and operators recognized a need to expand the capability of the autonomy engine. The *TIMED* version treated all DCB data points as 16-bit unsigned integers. While this was adequate for most cases there was a need for greater flexibility. On *CONTOUR*, the team enhanced the flight software implementation to support the use of 32-bit unsigned integers and IEEE-754 32-bit single precision floating point in the premise of rules. This provided far greater flexibility to the Safing Engineer in the development of the rules. Despite this increased capability, *CONTOUR* reused more than 90% of the design and implementation directly from *TIMED*. Furthermore, the changes were isolated to specific methods, and the API remained generally unchanged. This minimized impact to other subsystems within the software while providing greater capability.

7.3. *MESSENGER enhancements*

On *MESSENGER*, an even more powerful change was introduced: a Reverse Polish Notation (RPN) evaluation engine. Instead of restricting the rule premises to a list of predefined expressions, the RPN engine allowed any arbitrary combination of DCB data points, constants, and operators to be used as the premise of a rule. In addition, the list of possible operators found in rule premises was expanded. An entire new software implementation was required to perform the rule premise calculations using RPN. This replaced the older style premises with predefined expression formats. Despite this significant new development, the remaining elements of the autonomy implementation were directly reused from *CONTOUR,* including all of the management required to maintain persistence counters, fire commands and macros, etc. This implementation will be used on both *STEREO* and *New Horizons*. Again, a new technology and expanded capability was introduced while still leveraging the existing design and implementation from a previous mission (*CONTOUR*).

8. Plan for the future

The motivations for software reuse will only increase in the future. Where reuse enables JHU/APL to develop reliable flight software rapidly at low cost, new missions will pursue it. Two trends in particular are expected to affect the software development process and software reuse.

8.1. *New hardware platforms*

Only radiation-hardened microprocessors can be used for mission-critical applications in outer space. Development of these processors lags conventional processor development by several years; nonetheless, new generations of computer architecture become available to flight software developers and old generations become obsolete. For example, the BAE RAD750 processor (a radiation-hardened Motorola XPC750) will supersede the RAD6000.

Using newer processors affects the software development process in several ways. First, any time the processor or the tool chain changes at least some of the code becomes obsolete. JHU/APL's incremental reuse strategy is well suited to deal with these changes, because they can be isolated by phase and/or functional area and products not affected by the change can be reused. This is particularly true at the requirements level.

Second, because they are faster than older processors, newer processors typically perform more functions in their flight programs. This is likely to lead to fewer processors with more functions in each. For example, the G&C and C&DH functions can be combined into a single complex program on a single processor. The reused designs from individual programs can be combined in such a case; however, the more complex the software, the more ways that it can fail.

Third, more modern software tools are available on newer processors. These include more mature compilers following more modern language standards, full-featured debuggers, tools for visualizing processor performance, and stable operating systems. The use of these tools is likely to lead to a more reusable base of flight software products because fewer modules will be the result of customizing to meet the foibles of a less-mature tool.

8.2. *Unattended operation*

Increased spacecraft autonomy is perhaps the dominant trend in spacecraft flight software innovation today. Deep space missions must carry out their missions without oversight from the ground because the round-trip light travel time makes real-time remote operation impossible. For near-Earth missions the motivation for autonomous operation is cost: multi-mission programs continually strive to decrease the recurring cost of operations in order to fund the next generation of spacecraft.

TIMED has already made significant strides in the realm of unattended operations: operations of the instruments and the spacecraft are decoupled, and most ground contacts are now unattended. *STEREO* expects to achieve a similar level of autonomy during its operational phase. The next JHU/APL challenge will be two *Geospace* missions: Ionosphere-Thermosphere Storm Probe (ITSP) and Radiation Belt Storm Probe (RBSP) are part of NASA's "Living With a Star" program. They will doubtlessly reuse much of the base of requirements, design, and code developed for the sequence of missions described here in order to minimize operational costs.

9. References

[1] Malcom, Horace, and Utterback, Harry K., "Flight Software in the Space Department: A Look at the Past and a View Toward the Future", *APL Tech. Dig. 20*, JHU/APL, Laurel, MD, Oct – Dec 1999, pp. 522 – 532.

[2] Shumate, Ken, and Keller, Marilyn, *Software Specification and Design: A Disciplined Approach for Real-Time Systems*, Wiley, New York, 1992.

[3] *International Standard ISO/IEC 9899:1990: Programming Languages—C*, ISO, Geneva, 1990.

Appendix: Diversity of JHU/APL spacecraft missions

	TIMED	*CONTOUR*	*MESSENGER*	*STEREO*	*New Horizons*
Mission profile	2 years, 625 km Earth orbit	5 years, 1 AU heliocentric orbit	5 years transfer, 1 year Mercury orbit	2-5 years, 1 AU heliocentric orbit	15+ years, Solar System escape
Integrated electronic modules (IEMs)	2	2	2	1	2
Computers per IEM	1 C&DH 1 GPS Nav	1 C&DH 1 G&C	1 C&DH / G&C 1 Fault protection	1 C&DH 1 G&C	1 C&DH 1 G&C
Processor architecture	12 MHz Mongoose V (MIPS R3000)	12 MHz Mongoose V (MIPS R3000)	25 MHz RAD6000 (IBM RS/6000)	25 MHz RAD6000 (IBM RS/6000)	12 MHz Mongoose V (MIPS R3000)
Operating System	Nucleus+	Nucleus+	VxWorks	VxWorks	Nucleus+
Solid-state recorder	2.5 Gbit DRAM	5 Gbit DRAM	8 Gbit SRAM	8 Gbit SRAM	64 Gbit Flash
Power system	Movable solar array	Fixed solar array	Movable solar array	Fixed solar array	Radioisotope thermoelectric
Attitude control	0.05° (actual)	0.1° (planned)	0.02° (planned)	0.002° (planned)	0.002° (planned)
G&C sensors	Sun sensor, star trackers, gyros, GPS	Earth-Sun sensor, star trackers, gyros	Sun sensor, star trackers, gyros	Sun sensor, star tracker, gyros	Sun sensor, star trackers, gyros
G&C actuators	Reaction wheels, torque rods, solar array drive	Thrusters	Reaction wheels, thrusters, antenna, solar array drive	Reaction wheels, thrusters, antenna	Thrusters
Instrument data interface	1553 bus (all)	1553 (mass spec.) Dedicated (others)	Dedicated (imager) 1553 bus (others)	1553 bus (all)	Dedicated (all)
Downlink	10kbps – 4Mbps	10bps – 255kbps	10bps – 104kbps	11bps – 720kbps	10bps – 104kbps
Mission drivers	Decoupled instrument operations	Non-repeatable encounters, open-loop maneuvers	Mercury orbit thermal, non-repeatable encounters	Attitude jitter, decoupled instrument operations	Non-repeatable encounters, RTG, lifetime