Evaluating Flight Software Effort Estimation and Reusability Approaches for Planetary Exploration

Sarkis S. Mikaelian
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
Sarkis.S.Mikaelian@nasa.gov

Lloyd Manglapus
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
Lloyd.G.Manglapus@jpl.nasa.gov

Marek Tuszynski
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
Marek.W.Tuszynski@jpl.nasa.gov

Abstract*—Spacecraft Flight software is a critical component of every planetary mission. Besides the efficiency and cost-effectiveness of its development, proper planning efforts to maintain budgetary control, and adherence to the timeliness of schedule for cost-constrained missions are crucial for mission success. This is because there are initially many unknown factors such as technical uncertainties, resource availability, and mission requirements, making it extremely challenging to plan, estimate costs, and quantify the associated risks of developing novel capabilities and features. One solution for keeping costs reasonable and managing scope to mitigate these challenges lies in the adoption of software reuse and inheritance strategies. Software reuse and inheritance play a significant role in cost estimation as they offer reduced development cost, shorter development schedule, improved reliability, increased efficiency, and standardized consistency, ultimately contributing to more accurate estimations and enhanced management of the complex development process. As planetary missions become more frequent, and competitive, it is not feasible for each mission to develop architectures and infrastructure capability anew. However, successful reuse can be elusive and challenging given the distinct nature of each mission. The purpose of this study is to investigate the effectiveness of the approach taken by software management teams at NASA Jet Propulsion Laboratory (JPL) in modeling the cost of software upfront prioritizing the use of inherited flight software product lines that enable high reusability and ensuring reliability through proven flight records. We will examine historical trends of software reuse practices at JPL to highlight ways to maximize inheritance benefits across multiple planetary missions. Lastly, we will perform a quantitative analysis to assess the feasibility of a reusable software base, identify limiting factors by using the Europa Clipper flight software and its FSW Core Product Line (FCPL) as a baseline, and conclude the results. By quantifying the benefits of software reuse in cost savings, our findings will contribute to a comprehensive understanding of the efficiency and effectiveness of flight software reuse and inheritance for cost estimation. Moreover, the results of the analysis will inform and influence cost exercises, leading to risk-informed cost-effective decision-making for flight software effort estimation and inheritance considerations in future JPL missions.

TABLE OF CONTENTS

1. INTRODUCTION

As the demand for space exploration and scientific research in the field of planetary science continues to grow, the need for efficient and cost-effective flight software (FSW) becomes increasingly crucial. The development of robust and reliable software systems for planetary missions requires a thorough understanding of the factors that influence cost and efficiency, as well as the potential benefits of software reuse and inheritance. The issue of software reuse appears as a key cost driver in every flight project at JPL, where programs overstate inheritance plans and later suffer from cost overruns. These overruns often result from poor planning, lack of expertise, inadequate staffing, and imprecise cost estimation [13]. Flight projects at JPL often want to reduce their cost estimates by reducing development work and planning on inheritance, but it rarely works as expected as most inheritance assumptions are wildly optimistic. Also, reuse is not well understood, and it is hard to quantify success even for software professionals. When inheriting code, integration typically requires a certain degree of source code adaptation, which incurs an anticipated programmatic cost. This adaptation often proves more difficult to estimate than new development. Occasionally, projects spend more time modifying inherited code than originally planned as a result of avoiding the cost of developing it anew. There are two distinct approaches to inheritance (Clone & Own and Use-as-

is), the assumption of more inheritance, does not necessarily mean less cost as projects often Clone and Own, and spend more time tailoring reusable software. Modifying code is slower, more error-prone, and can be more difficult than writing new code. There is a fine threshold before reuse becomes inefficient and less desirable compared to maximum inheritance because once a module requires 50-60% modification, extensive modifications are treated as new code.

The scope of this paper focuses on flight software cost estimation, reuse, and inheritance for planetary missions at NASA JPL. The research examines the definition of spacecraft flight software reuse and inheritance according to their implementation at JPL, reviews heritage flight software and its influence on Europa Clipper (EC) FSW (ECFSW), and investigates JPL software product development and reuse trends. Then, the paper employs a quantitative sensitivity analysis using the COCOMO-based SCAT model [7] for a specific set of alternatives and scenarios and compares the results in a comprehensive comparative analysis – described in detail in the Methodology section. Finally, it concludes, discusses findings, and offers guidelines and insight for considering alternatives and maximizing the benefits of reuse for cost savings according to the identified key sensitivity factors.

The high-level objective of the analysis is to examine the success and feasibility of a reusable software base, such as FCPL, using Europa Clipper as a baseline. The analysis aims to investigate the basis of FCPL to identify effective software reuse practices and the sensitivity of inheritance rate thresholds to maximize reuse. Quantifying the benefits of effective reuse and utilization of software product lines helps reduce the risk of software-related issues and failures contributing to the overall success and reliability of future missions [13]. The goal is to identify key cost drivers and reuse limitations that could influence future flight projects such as Mars Sample Return (MSR) – Sample Return Lander (SRL), and make recommendations for improved decision-making and accurate cost estimation. A secondary objective is to concentrate on the FCPL layer of ECFSW by isolating its development cost and effort in the analysis and estimating the cost penalty of adopting its development as a reusable product within the scope of Europa Clipper. This is due to FSW Core's extensive heritage expectations at JPL [17] and the high probability of inheritance on MSR/SRL. The results are expected to demonstrate its reusability and inheritance nature, considering that FCPL is an FSW product with many development instances and a continuously ongoing effort toward a final, stable, and possibly purely reusable version. The paper will only cover the missions described in Table 1, which enable the analysis of FCPL and FSW cost estimation for reusability and inheritance.

Table 1. Mission Description & Objectives

Mission	Objective
Europa Clipper*	Explore Jupiter's icy moon Europa, investigating its potential habitability and seeking signs of past or present life by studying its subsurface ocean, ice shell, and geologic features using a suite of scientific instruments [4].
Psyche	Investigate the intriguing metallic asteroid 16 Psyche, located in the main asteroid belt, to understand its composition, structure, and potential insights it offers into the violent collisions that shaped our solar system's history [12].
Mars Sample Return	A groundbreaking, multi-mission endeavor that involves collecting and caching Martian rock and soil samples with the Perseverance rover, launching them back into Mars orbit, and finally returning them to Earth for in-depth scientific analysis to help unravel the planet's geological and potential biological history [9].

(*The research focuses on Europa Clipper and only references Psyche and Mars Sample Return as needed**)**

2. JPL FLIGHT SOFTWARE EVOLUTION

Flight Software Inheritance & Reuse

Flight Software inheritance and reuse are often used interchangeably at JPL, however, according to [5] there is a distinction between them that corresponds to JPL's initial scope of flight software inheritance and reusability definitions. Flight software inheritance refers to the process of building upon existing software frameworks, structures, or components to create new applications, functionalities, or systems for aerospace projects. Inheritance involves leveraging the underlying principles, design artifacts, and architectures of existing flight software to streamline development, reduce development costs, and maintain system reliability. Flight software reuse, on the other hand, involves the use of existing software drivers, components, modules, and subsystems in the development of new flight software systems. Reuse can be accomplished through various methods, such as the use of libraries, frameworks, or middleware solutions, which provide pre-built functionality that can be easily integrated into new software systems. The primary goal of reuse is to minimize the effort required for software development by leveraging proven, tested, and reliable software components which can result in significant cost savings, reduced development time, and improved software quality and reliability. Furthermore, the overall objective is to maximize the utilization of both methodologies by adopting practices that promote inheritance and reuse, optimize their software engineering efforts, and maximize the value of their software assets.

Historically, JPL has been excessively optimistic in its estimates for flight software reuse since reuse estimates have driven development budgets by lowering risk and contributing to reducing flight software development cost and verification and validation (V&V) efforts [13]. Traditionally, JPL exercises rigorous inheritance activities through detailed system characteristics and capabilities consideration and inheritance reviews during the design and architecting phase of the flight software lifecycle as seen in Figure 1. The process starts by drafting an initial estimate with a certain degree (percentage) of reuse and inheritance assumptions in SLOC which includes design artifacts, architecture, requirements, and source code, and refining it through the PDR and CDR phases. It is important to note that there are typically two different approaches to inheritance: (1) Clone and Own, and (2) Use-as-Is/Pure Reuse.

"Clone-and-Own" is known as traditional inheritance where an existing software component, module, or subsystem is copied and then customized or modified to meet the requirements of a new project or application. Once cloned, the owned software becomes a separate entity from the original and is isolated from updates or changes to its original version. This approach provides faster initial development, and flexibility to tailor the software to specific requirements and needs not met by the original version. On the other hand, it increased maintenance efforts as patches and updates are not automatically applied to the cloned version and may cause inconsistencies and compatibility issues in the future because of its divergence.

Pure Reuse is a much-advanced approach that involves using an existing software component, module, or subsystem in its current form, without modification, and directly integrating it into the new system. It is widely used among aerospace and defense contractors to produce larger-scale FSW and adopt modular design practices to make modules as reusable as possible. This approach offers reduced development time and effort when reusing without modification, and lower maintenance costs since updates and bug fixes made to the original software are propagated to all systems using it. However, it has limited flexibility as the reused software may not fully address the specific requirements of the new project, and may introduce risks related to compatibility, support, and licensing due to reliance on external dependencies.

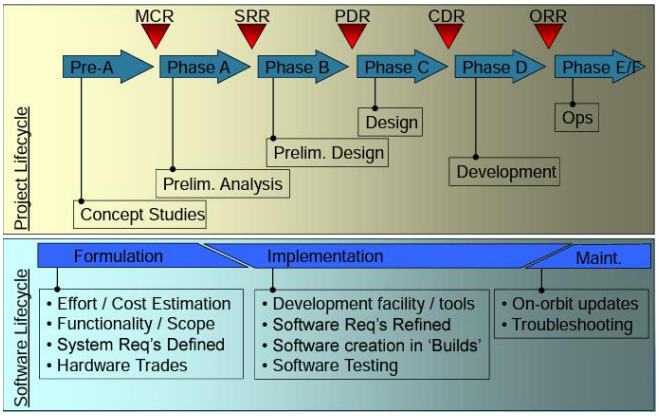

Figure 1. FSW Development Lifecycle [5]

"Clone-and-Own" is mostly adopted when inherited codebases have not been designed for proper reuse. When cloned and owned, a project can lower risk, but the process may be very challenging without acquiring developers familiar with the software and systems. Therefore, the availability of the original developers is an important factor considered in inheritance decision-making (e.g., Mars missions).

JPL Heritage Flight Software & Core Product Line

Historically JPL's heritage flight software consisted of many tightly coupled modules integrated into a monolithic delivery to a specific flight system and mission. Inheritance in terms of pure reuse was not an objective [8], JPL followed and still follows a "Clone and Own" model of FSW reuse, inheriting code from a previous project and adapting it to suit the next project's specific needs. This approach has limited reuse across projects and attempts to scale a particular architecture beyond its original intention. According to data from the ESD Mission Software: State of Software Report 2018 [13], JPL has been pursuing several ways to increase reuse for in-house projects including the development of the Flight Software Core Product Line (FCPL). MSL challenged the "Clone and Own" development approach and pushed the boundaries of architectural complexity when the FCPL concept was introduced in FY2013 as an institutional product line designed for multi-mission reuse. In 2013/2014, JPL officially began developing a software product line called "JPL Flight Software Product Line" [17] with elements inherited from MSL and SMAP. The diagram in Figure 2, details the traceability of the FCPL roots to MSL and SMAP which had their inheritance lineages traced back to Pathfinder.

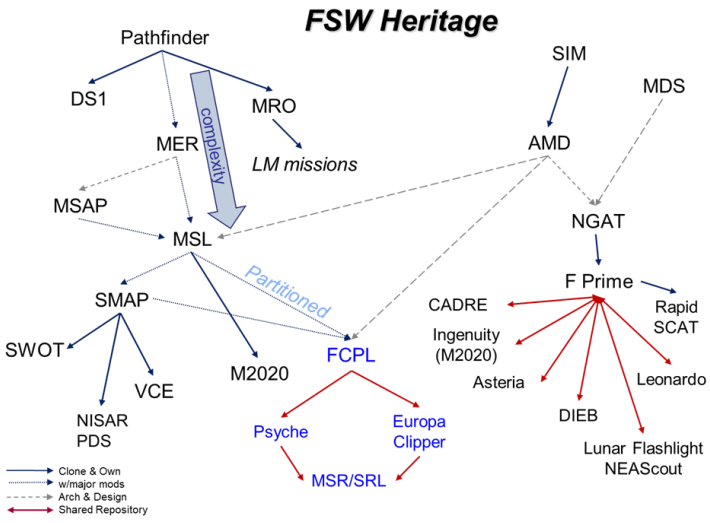

Figure 2. JPL FSW Heritage Lineage [8]

Initially, an avionics Reference Bus was established as a key hardware framework to provide common and essential services such as compute elements, communication, data management, and device interfaces to support payloads [8]. This effort aimed to develop implementation templates with built-in flexibility for future configurations of in-house or

mixed-mode mission applications that accommodate different reference bus configurations of FSW implementations. Then the Reference Bus commissioning led to the development of the JPL FSW Product Line to:

1. Facilitate the delivery of integrated avionics with FSW products, in addition to distributed development and delivery of FSW capabilities to projects.
2. Improve the reuse potential of flight software while leveraging existing flight-proven implementation approaches and assets.
3. Eliminate or reduce start-up time and cost for flight software infrastructure development, enabling new projects to proceed directly to mission-specific application implementation.

As shown in Table 2 of Figure 3, the early version of this product included a "Core" component – a collection of reusable FSW capabilities available for deployment on a wide range of flight projects. This offered an FSW product development strategy that includes architectural guidelines and well-defined interfaces so that projects can integrate mission-specific software functions into existing assets to create a complete FSW system. It also enabled a management approach where the Product Line assets ownership is allocated to the Line Organization, which provides configuration control and oversees proposed use, deployment, maintenance, and evolution prospects.

Later, the product was refined, and an improved version of the JPL Product Line called the Flight Software Core Product Line (FCPL) was established to support large flagship-type missions [8]. The new product was to further develop the reusable FSW assets to include a reference FSW architecture, infrastructure requirements and components, development processes, unit and integration tests, and utility tools. Similar to its predecessor, its refined objective was to:

1. Define FSW principles and architecture suitable for highly complex systems.
2. Break the Clone-and-Own cycle by developing a reusable Core FSW that implements and forms the basis of the FSW principles and architecture.
3. Componentize FSW to move away from monolithic executables, partition software into components – a collection of one or more tasks providing well-defined functions and interfaces that operate within a defined address space context, adopt Space and Time Partitioned FSW architecture with multiple components, and employ operational improvements to manage components versus whole.

The overall development of FCPL encompassed 4 major milestones that extend until today:

- In 2014, a reference FSW architecture and Core FSW requirements were defined, which led to the identification of the Core FSW modules. The implementation of these modules began shortly after they were defined as one Core component. Lastly, the FSW was prototyped with a Space Partitioning architecture which included a Core and Demo component on the VxWorks 6.9 RTOS – Real-Time Process (RTP) instance.
- In 2015, the RTOS trade was completed which meant transitioning to Green Hills Integrity OS. Then prototyping FSW Core and Demo Component with Space and Time Partitioning was finalized and successfully demonstrated the FSW execution on the RAD750 testbed utilizing Space and Time partitioning, Command and Telemetry, and compatibility with the ground system.
- In 2016, dynamic loading of FSW components and shared libraries was introduced along with 2 new features, a SHELL component, and an execution model – rate groups. This was followed by a successful demonstration of Core FSW 1553 Bus Control and the prototyping of Europa Clipper risk-

JPL FSW Product Line Core Component
Core Component Functional Domains
Configuration Management
Command
Telemetry – EHA, EVR, Data Products
File System
Parameters
Intra/Inter-Component Messaging
Operating System and Computer Hardware Abstraction
Timekeeping
Utilities (i.e., math, compression, bit/byte ops, etc.)
Health
IO

Table [2] – JPL FSW Product Line Core Component

FSW Core Product Line (FCPL) Components	
Partition	**Functions**
Kernel	Platform Services – running in Supervisor Privilege mode: configuration, health, memory scrubbing
Platform	Platform Services – discretes, configuration, health, autopsy, resource tracking, memory management, messaging, state database, utilities, update, time, initialization
IO	1553 Bus Control, 1553 Remote Terminal, REU management, Cross String
Uplink	Uplink, Command, Critical Relays, Sequencing
Downlink	Downlink, Channelized Data, Event Reporting
Data	Engineering File System, Parameters, Records, Dataproducts, CFDP
Shell	Console Services

Table [3] – JPL FCPL Components

Figure 3. JPL FSW Product Line Core Component Decomposition to FCPL Components [16]

reduction devices – Propulsion, Power, and Remote Engineering Unit FSW components.
- In 2017-2018, FSW Core was decomposed into 8 smaller components, a new CPU Performance Measurement and Reporting feature was added, and the benchmarking of representative FSW build and FSW optimization were initiated. Lastly, the FCPL planned deployment to Europa Clipper as part of Europa FSW, and Psyche for integration into Psyche FSW.

As of 2018, the FSW Core or FSW Core Product Line (FCPL), has been a subset of the ECFSW intended to be reusable to other deployments of the Avionics Reference Bus architecture, initiated under ESD institutional funding but has been in development under ECFSW for Europa and other projects. FCPL is currently deployed on the primary flight computers of Europa Clipper and Psyche missions. It inherently employs Time and Space Partitioning (TSP) architecture [11] at the core of its functionality which is one of its main design implementation drivers on both Psyche and Europa Clipper. It is safe to say that it has achieved componentization of 7 Core components, decomposed from the initial single "Core" component as shown in Table 3 into ~50 Core modules, multi-mission future use-readiness, unique hardware configuration, mission needs optimization and configuration management. Compared to its early version *R4.10* with 49 modules [13], on Europa Clipper, the current FSW version, *R10.2* consists of a set of 46 common and reusable software modules described in Appendix C – deployable in future missions to be utilized as a starting point for their FSW development.

Inheritance & Reusability Trends at JPL

Most flight software costs are labor-based, with developers generating code and testing the software, besides some costs required for hardware and software development tools such as compiler seats, software licenses, configuration management software, etc. [10]. Cost and effort for flight software at JPL are usually estimated through professional judgment based on subject matter expertise in the field of Software Engineering, then supplemented and refined by data produced by tools such as SCAT, COCOMO, Monte Carlo Sizing, and COSYSMO 2.0. Another popular method JPL relies on for FSW cost estimation is TeamX Concurrent Engineering Teams which mostly focuses on early lifecycle design which involves using a COCOMO-based modeling tool during project proposal and formulation [3] to verify the estimates by cross-comparing the outcomes to get consistent results. Over the years, JPL has established rigorous practices such as the JPL Reuse Program [2] – one of the early reuse attempts with an aim to develop a quantitative understanding of the factors that encourage or inhibit software reuse and reuse-informed productivity improvements by utilizing the model described in Figure 4 to keep track, collect data, and quantify software reuse exercises across the lab. Other programs such as the Software Quality Improvement (SQI) project [13] have been very proactive in studying institutional software practices and producing comprehensive periodic reports [13] which have been extremely important for cross-comparing reuse and cost estimation.

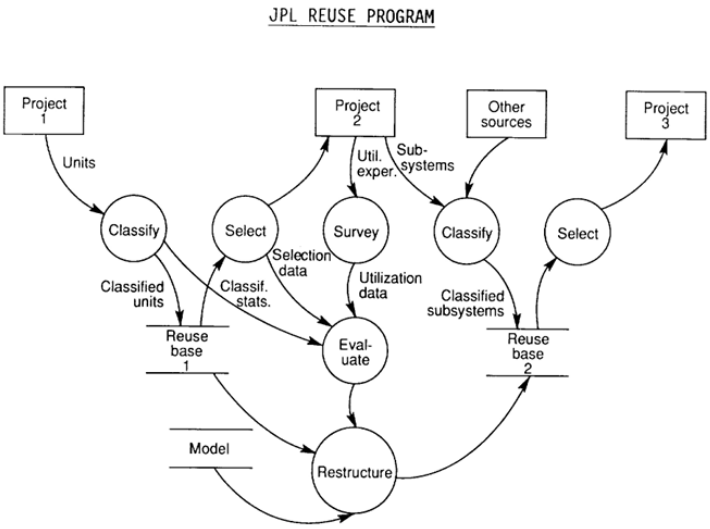

Figure 4. JPL Reuse Program Model [2]

According to the ESD Mission Software: State of Software Report 2018 [13], over the past 10 years, JPL had many efficient and successful reuse cases and was able to demonstrate a 47% reuse rate for in-house built spacecraft flight software – shown in Table 6. The first was the Soil Moisture Active Passive (SMAP) project's 32% FSW reuse with almost accurate work months and SLOC estimates shown in Table 4. Then, both, InSight and Orbiting Carbon Observatory 2 (OCO-2) were able to achieve extremely high reuse rates (> 75%). InSight was launched using 88% reused software and OCO-2 was rebuilt from the failed predecessor (OCO-1) with almost 100% reuse. Another recent example of maximizing reuse is the Mars 2020 project, reflected in Table 5, which reused 56% of the flight software infrastructure inherited from MSL for its surface FSW and 87% for cruise/EDL, the highest for any JPL Mars mission and of its scale.

Table 4. SMAP Mission FSW Estimates [14]

SMAP	Initial Estimates			Actual
	Initial	PDR	CDR	Completion
WM	876	876	876	932
SLOC	180,000	180,000	180,000	158,379
Assumed Inheritance	83%	83%	83%	32%

Table 5. Mars 2020 Mission FSW Reuse Rates

Event	Percent New	Percent Unchanged	Percent Modified
Cruise/EDL	6%	87%	7%
Surface	26%	56%	18%

Since 2009, JPL has had an upward trend for its effective flight software development practices and processes. Though not pure reuse, one of the best-known cases that enclose a product line–like flight software is the Electra originally designed and used on Mars Reconnaissance Orbiter (MRO) and since then it has been adapted with a redesign and reused on multiple missions such as M3, MSL, MAVEN, and TGO [13]. Nevertheless, the data in Table 6 shows that software inheritance rates have decreased since 2014 but JPL has been trying to reverse this trend and increase reuse ever since by encouraging the development of in-house FSW Product Lines such as FCPL and F Prime [1].

Table 6. Reported Reuse Rate Trends at JPL (2009–2018)

Reuse Level	Percent Per Year			
	2009	2011	2014	2018
No Reuse	45%	36%	38%	42%
Less than 25%	20%	17%	4%	19%
25–49%	10%	7%	4%	13%
50–74%	6%	9%	8%	11%
75–99%	19%	33%	30%	15%
Average Reuse Rate*	26%	37%	34%	47%

JPL systems have been getting bigger by the mission, in scope, size, instrument count, mass, and complexity. Consequently, FSW system size has been increasing since the 1980s as the collected software size data measured in SLOC in Figure 5 indicates a steady increase in the size and complexity of JPL's software systems over time. This calls for effective and advanced reuse practices since bigger systems require larger software with more code, which means better opportunities for cost saving. Also, if done right, pure reuse can provide an opportunity to focus on developing new system functionalities – features once novel, to become the new baseline.

When software can be reused, it can lead to cost savings and higher reliability. However, when overly optimistic assumptions are made, it increases the original cost and lowers the quality of the final product more than it would have been had it been planned with realistic assumptions. The most recent planned inheritance includes the Psyche and Europa Clipper FSW as adaptations of the JPL Core reusable flight software with high heritage and major reuse of subsystems from SMAP and MSL including the Attitude Control Subsystem (ACS). However, the actual implementation for both has diverged from their initial plans, resulting in two variations of FCPL during the development of FSW version R4.10. This divergence was partly due to the fact that the inherited code was not designed for reuse, alongside other integration issues and constraints. Nevertheless, Psyche became the first mission to operate using JPL's FCPL common core flight software, utilizing capabilities that enabled new levels of onboard autonomy [22].

An alternative to reuse that is a commonly adopted cost reduction strategy is auto-generated source code that is developed by providing models and high-level abstractions of software functionality and behavior to an auto-coder engine that translates the input into software source code. However, this may increase testing costs and complicate debugging and maintenance efforts because of code bloat and concerns about the quality of the software produced since auto-generated source code is larger than manually written code that provides the same functionality. For reference, ECFSW's total logical SLOC versus auto-generated SLOC for R10.0 has a ratio of 1.1:1 – shown below in Table 7 and is factored into cost estimation.

Table 7. Europa Clipper FSW (R10.0.0) SLOC Statistics

Logical SLOC	Hand-coded C Code	Auto-generated C Code
	366,930	330,934

Contrary to Flight software, ground software systems at JPL have higher reuse rates [13] since most of the modules and functions are used with no modification from one project to the next. While flight software does not enjoy the same stable environment due to many complexities, which is why realistic reuse is much more difficult to achieve in flight missions. A very successful instance of this with a reuse rate of almost 100% is the Advanced Multi-Mission Operations (AMMOS) ground software [13] that was designed and developed to be highly reusable, with built-in adaptations for ease of reuse such as Mars, Jupiter, and other planetary destinations. AMMOS includes software for planning and sequence, up-linking commands, navigation, telemetry downlink, science data processing and archival tools, and miscellaneous administrative tools. The objective is to reuse this functional suite instead of individual projects developing their own ground system, and then tailoring the software to mission requirements and operations. Similarly, a high reuse rate of 98% was achieved by another JPL mission ground systems product called Multi-mission Ground Systems and Services (MGSS) [13].

Based on various reports, JPL seems to generally follow established processes, as the process performance for most Class B and large Class C development tasks have been at or above 80% [13]. Since 2007, JPL has been one of the first NASA centers to achieve a CMMI Level 3 ("Defined") rating for its flight software process maturity, according to the Capability Maturity Model Integration (CMMI) [19], and has sustained it every 3-year assessment since then. The CMMI process improvement program consists of five Maturity Levels and is considered to reflect best practices for efficient software engineering management and oversight. It ranks optimal practices for effective flight software development,

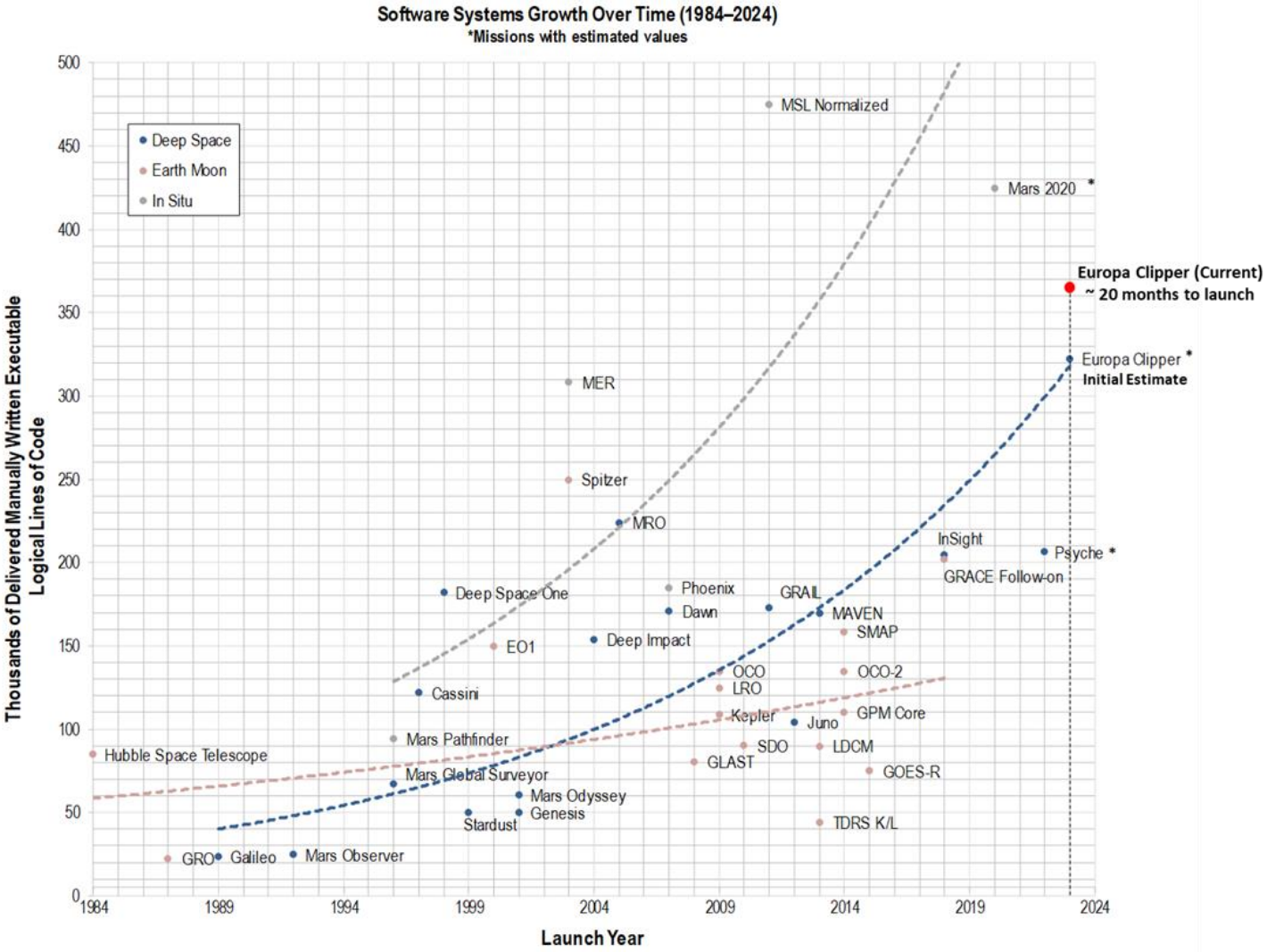

Figure 5. JPL Software System Growth Over Time [13]

including software reuse and inheritance across select planetary missions. These practices in turn influence cost estimation and map directly to JPL's COCOMO-based SCAT model's Process Maturity (PMAT) scale factors input parameter [7].

Most recently in 2022, JPL reattained a Level 3 status, during which the appraisal focused on Class-B Flight Software and covered findings from SQI, SQA, SWOT, MAIA, MONTE, NISAR, Mars Sample Return (MSR), VERITAS, Mars 2020, Psyche, and Europa Clipper Mission software [19]. This designation acknowledges that the organization is proactive, its processes are characterized, and projects tailor their procedures based on the organization's standards. Although there is room for improvement, JPL's advanced rating is a strong position, demonstrating the center's adherence to processes and coherent development plans that extend beyond the uniqueness of JPL's software environment.

3. EUROPA CLIPPER FLIGHT SOFTWARE

The Europa Clipper spacecraft flight software is a class B Non-Human Space Rated Mission Critical/Safety Critical software system written in the C computer programming language that executes on the BAE RAD 750 flight computing platform using the Green Hills Integrity Real-Time Operating System (RTOS). It serves as the central control center of the spacecraft and is responsible for communicating with Earth, collecting data from all peripherals, and controlling them to achieve mission objectives. The software manages subsystem peripherals, including avionics hardware, telecom, power, propulsion, mechanisms, thermal, guidance, navigation, and control (GNC) devices, and payload instruments. Its primary functions encompass uplink, commanding, sequencing, onboard data handling, instrument data collection, compression, telemetry, and downlink. Additionally, the software handles vital tasks such as science avionics management, timekeeping, file systems, power management, thermal control, and payload management. It also includes higher-level behaviors and autonomy, such as launch, Jupiter orbit insertion, automatic maneuvering, and fault protection.

A detailed description of ECFSW's main functions can be found in Table 8, below:

Table 8. ECFSW Main Functions

Core Infrastructure	Application Functionality	Instrument Management
1. FSW Infrastructure **2.** Partition Management/OS **3.** Health Monitoring **4.** Uplink **5.** Downlink **6.** Timekeeping **7.** Memory Management **8.** Commanding **9.** Sequencing **10.** Channelized Telemetry **11.** Event Reporting **12.** Data Products **13.** Bootup and Initialization **14.** Cross-String Arbitration and Communication **15.** REU and shared records **16.** Utilities, Compression **17.** CFDP **18.** File System **19.** Parameter and Records Service **20.** Shell Service	**1.** Avionics Hardware Management 1553 Bus Management **2.** Measurement Collection, Conversion, Distribution **3.** Power Switching **4.** Thermal Control **5.** Mode and System Configuration Management **6.** Fault Management: Monitors, Responses, Behaviors **7.** Redundancy Management **8.** Data Storage **9.** FSW Load & Patch **10.** GNC device management, attitude control, ephemeris, and motion behavior **11.** Europa Behaviors: Launch and JOI **12.** Telecom: Radio Management, Com Behaviors **13.** Deployments	**1.** Bulk Data Storage, Recording, Filtering, BDS **2.** Instrument Common Framework **3.** Europa Clipper Magnetometer Instrument Manager **4.** Europa Imaging System Narrow Angle Camera Instrument Manager **5.** Europa Thermal Emission Imaging System **6.** Mass Spectrometer for Planetary Exploration **7.** Mapping Imaging Spectrometer for Europa **8.** Plasma Instrument for Magnetic Sounding Lower Instrument Manager **9.** Plasma Instrument for Magnetic Sounding Upper Instrument Manager **10.** Radiation Monitor **11.** Radar for Europa Assessment and Sounding **12.** Surface Dust Mass Analyzer **13.** Ultraviolet Spectrometer

Flight Software Architecture & Design

Embedded Flight Software is critical for mission success in flight projects, it enables spacecraft hardware, science instruments, and flight components to operate as an integrated on-orbit or ground science observatory. It is also expensive to develop and test in parallel (V&V). Embedded software further complicates the typical software development lifecycle which in turn complicates cost and schedule as it requires rigorous V&V, deterministic real-time performance for attitude control, high-speed data, crucial flight hardware integration, high support autonomous operations, remote maintenance and system updates, and high reliability and fault tolerance exercises to ensure spacecraft safety during failures [11]. Consequently, all these requirements translate into cost factors that dictate the overall FSW cost of a mission.

In the past, JPL's most initial estimates of software reuse have been unrealistically optimistic for taking high heritage from previous missions [20]. However, Europa Clipper did not assume any code inheritance, it only inherited as little as 5% from the design of certain aspects of heritage core FSW. Instead, it utilized FCPL and developed it into a TSP-based architecture – previously discussed and described in Figure 6. The FSW was divided into partitions mapped to a combination of mission-specific components that provide vehicle and instrument functionality, and Core FSW components that provide all common multi-mission infrastructure functionality.

The Time and Space Partitioned system [11] employs proven MSL and SMAP heritage design and code as a basis (demonstrated in Figure 2) for its implementation of a scheduled and resource-partitioned reusable product that provides C&DH compatible spacecraft infrastructure instead of a priority pre-emptive system for improved reliability, determinism, and manageability. This component-based architecture is consistent with long-term organizational technology roadmaps. It preserves functional constructs from Pathfinder to M2020 but is yet novel to JPL since it transitions from an event-driven priority-based system to scheduled, polling execution-handling of events, and from real-time interrupt (RTI) preemption jitter to deterministic partitioned timeline delays and latency management [20]. The componentization is achieved by grouping several

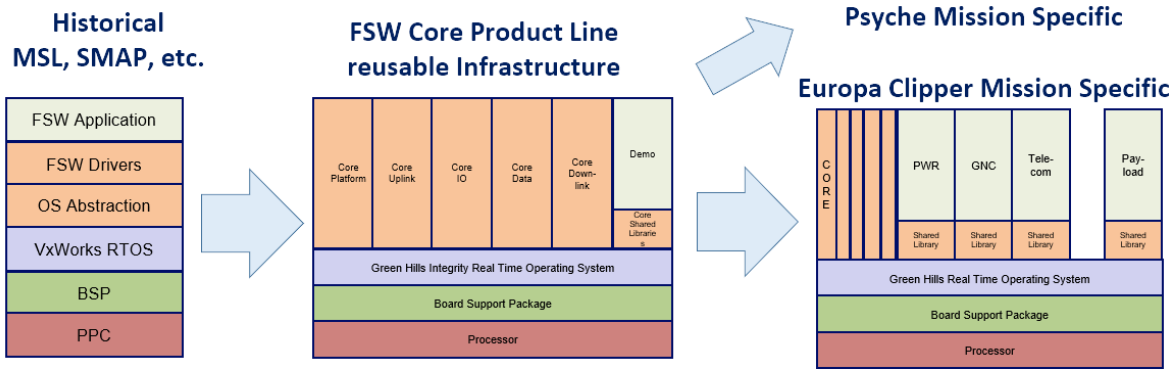

Figure 6. FCPL Architecture [16]

software modules as shown in Table 9 of similar criticality, logical relationships such as GNC functions and Thermal control functions, and common external client interface needs into 14 components – Tables 3 & 9, to minimize data flow across partitions considering performance and resource impacts.

The architecture assumes major design inheritance instances from heritage flight software including:

1. The Engineering File System (EFS) from the FSW functionality list described in Table 8, MSL heritage-based hierarchical directory structure for file-based storage and retrieval with ground commands and telemetry, and FSW interfaces. Code derived from the MSL flight file system used on MSL, M2020, Psyche preserves heritage system behavior and tailors to the responsibility of storing all on-vehicle files and diagnostic products that are not directly sourced from the instruments.
2. SMAP heritage-based Attitude Control Subsystem (ACS)/GNC that provides an abstraction of FSW methods from GNC algorithms through the isolation of inputs and outputs into one key structure that simplifies integration efforts.
3. CCSDS File Delivery Protocol (CFDP) component from Goddard Space Flight Center's (GSFC) cFS [18], which enables reliable file transfer over a noisy and interrupted communications link, allows the receiving entity requests retransmission of missing data and includes no-handshake simplified mode with additional tailored usage to Europa Clipper's needs.
4. Partial-heritage Fault Monitor design pattern from MSL in which local monitors link to a local response, and a system monitor links to a system response but optionally links to a local response. The pattern was tailored to ECFSW with the addition of a pre-condition as an explicit step in the fault monitor evaluation and the branch that triggers the execution of a local and system response during the same monitor evaluation cycle [16].
5. Heritage-based State Chart Autocoder (SCA) that is used to generate flight code framework for Hierarchical State Machines from MagicDraw UML/XML files, and for Linux-based test harness framework [16].

Table 9. Europa Clipper FSW Components

Partition	Function
System	System Fault Protection, Modes, and Behaviors
GNC	GNC Device Management (IMU, SRU, RWA, DSS), GNC Controls, Propulsion (PME device management)
Power	Power Device Management (PCDA)
Thermal	Thermal Device Management
Telecom	Radio Management, Radio Space Wire Device management
Instruments	Instruments Management
Bulk Data Storage	Bulk Data Storage Management

Time & Space Partitioning

In flight software, Time and Space Partitioning (TSP) architecture is a type of system architecture used in safety-critical systems that involves dividing the avionics system into multiple partitions, where each partition is allocated a specific portion of the system's resources, such as memory, processing time, and I/O bandwidth [6]. ECFSW achieves this by using Green Hills Software INTEGRITY 11.4.4 RTOS which provides Time and Space Partitioning capabilities and utilizing the FSW design to take advantage of flight hardware partitioning characteristics.

For time partitioning, the RTOS guarantees each component's slice of processing, it divides the system's processing time into discrete time slots or periods and allocates each component a specific time slot during which it has exclusive access to the system's resources. The size of each partition window is set at design time, and it is based on the CPU utilization of the task(s) that the partition has to perform [21]. The scheduling is time-preemptive – threads running in a time partition are preempted when the execution of the time partition reaches the end of its time allocation. One Major Frame repeats at a defined rate of 1 second (1 Hz) during which 32 identical Minor Frames (32 Hz), subdivided into fixed time partitions executed in a repeating sequence of 4 Minor Frames eight times (8 Hz) to complete the 1 Hz Major Frame. Each FSW component is allocated to one time

partition where its thread execution only occurs within the bounds of its time partition.

For space partitioning, the FSW uses shared memory constructs to enable efficient data access between multiple components, and device register and memory spaces that are specifically mapped only to those components that are designed to access them. The space partitions in ECFSW have virtual address spaces (VAS) that are managed by the RTOS and CPU Memory Management Unit (MMU) to divide the system's memory and I/O resources into discrete regions [21]. The RTOS and MMU allocate each component into one space partition in which the component accesses its data within that VAS and cannot access data in other components unless explicitly shared or mapped to its virtual address space. An out-of-bounds memory access attempt raises a configurable error response that results in the suspension of the executing thread or system reset. This guarantees one VAS cannot corrupt another and that each partition has exclusive access to its allocated resources and cannot interfere with other partitions.

The TSP architecture on Europa Clipper was designed to provide a high level of fault tolerance and safety for its safety-critical flight software and avionics systems. The isolation of each partition in the architecture ensures that a failure or violation in one partition does not affect the operation of others as FSW attempts to resolve them through local and/or system fault protection mechanisms. This allows the system to continue functioning in the presence of faults and errors, with system-level design exceptions such as fatal reset responses for non-recoverable faults encountered in a partition.

4. ANALYSIS METHODOLOGY

This analysis assumes the scope of the Europa Clipper spacecraft flight software metrics to be specific for the software that runs on the Europa Clipper Compute Element (ECE) hardware, developed by the avionics flight software team. These metrics do not encompass simulations, ground data, ground support, bench tests, or any other support software. Furthermore, they exclude flight algorithms developed by the GNC team, software executing on instrument hardware, frontier radio, and peripheral devices such as IRU, SRU, or RWA. The Software Quality Assurance (SQA) effort, which falls under the Mission Assurance organization, is also not covered by these metrics. Also, the source lines of code (SLOC) counts, and effort calculations do not include the Integrity RTOS or any vendor libraries, such as BSP or Lua Shell.

SCAT/COCOMO II Model

The COCOMO model is a widely used software cost estimation model. The model estimates software development effort and cost in work-months (WM) based on the size of the project (measured in Source Lines of Code SLOC or Function Points) and several cost drivers, which represent parametric factors that can influence the cost of FSW, such as size, inheritance, personnel capabilities, project complexity, and development environment.

The SCAT, or Software Cost Analysis Tool used in this study, is an Excel-based adaptation developed by JPL, of the University of Southern California's Constructive Cost Model II (COCOMO II), which has been validated using real data, providing reliable flight and ground software estimates within the JPL environment [7]. SCAT enables the use of a range of values (Low, Most Likely, High) to account for uncertainty over 35 adjustable parameters, generating probability distributions through Monte Carlo techniques. In contrast to the USC version of COCOMO which offers a single-point estimate, SCAT produces distributional estimates in the form of cumulative distribution functions, or "S-curves." We use this tool to supply software cost estimates in Work Months by combining Monte Carlo simulation with COCOMO II cost drivers and algorithms to perform further sensitivity analysis. The model is composed of four main components – Size, Reuse Parameters, Effort Multipliers, and Scale Factors that result in 35 different modifiable parameters which will be used to obtain an effort estimate, as described in Appendix F.

Sensitivity Analysis

One of the main causes of cost growth is overly optimistic assumptions for inheritance and reuse. A major disadvantage of reuse is overestimation which results in a final product more expensive and of lower quality than it would have been under realistic planning and a set of assumptions. This section employs a quantitative sensitivity analysis using the COCOMO-based SCAT model [7] for a specific set of alternatives and scenarios to identify key sensitivity factors that influence FSW cost estimation and test the sensitivity of inheritance rate thresholds to maximize reuse. By employing this methodology, we wish to explore and compare the impact of various scenarios by changing some of the input parameters that have been identified as initial sensitivity factors and verify their impact on the outcome in many iterations of the results. For example, when we wish to explore the impact of inheriting reusable software from previous projects, we run a SCAT estimate using a zero initial adapted SLOC, then run another SCAT estimate using a few thousand initial adapted SLOC to measure the impact by comparing both estimates which will be a decrease in cost [7].

The goal of this activity is to develop a quantitative understanding of the factors that encourage or inhibit software reuse and of productivity improvements achievable through reuse in the flight software application domain of Europa Clipper. The primary activity is the measurement of parameters relevant to reuse in a constant ECFSW environment. In order to achieve the analysis objective:

1. We develop a baseline SCAT model to allow the assessment of competing reuse techniques on the project level.

This includes the current/actual parameterization of ECFSW (as of R10.0.0) with all of its software modules, actual reuse rates, and logical SLOC - scenario 1 in Table 11.

Then, repeat the process 4 more times with different assumptions:

1) Assume the initial estimates parameterization of ECFSW with all of its software modules, estimated reuse rates, and logical SLOC - scenario 2.
2) Assume an estimated parameterization of ECFSW with all of its software modules, estimated reuse rates, and logical SLOC at PDR - scenario 3.
3) Assume an estimated parameterization of ECFSW with all of its software modules, estimated reuse rates, and logical SLOC at CDR - scenario 4.
4) Assume an estimated parameterization of ECFSW with all of its software modules, actual reuse rates, and projected logical SLOC at completion - scenario 5.

2. Extend the reuse consideration from the initial baseline rates to assume pure reuse of all FCPL modules - scenario 6.

   Then, repeat the process with the assumption of a partial "Clone-and-Own" inheritance approach - scenario 7.
3. Expand on the baseline implementation in scenario 1 to assume no effort (LOW) was spent to make the product reusable - scenario 8.

   Then, repeat the process with the assumption of very high effort spent to make the product reusable across the product line - scenario 9.
4. Expand on the baseline implementation in scenario 1 and extend the reuse consideration rates to assume 50% pure reuse of all software modules - scenario 10.

   Then, repeat the process with the assumption of 70% pure reuse of all software modules - scenario 11.
5. Finally, develop a SCAT model with the current parameterization of ECFSW's FCPL software modules with no reuse, actual logical SLOC, and nominal effort spent to make the product reusable - scenario 12.

   Then, repeat the process with the assumption of very high effort spent to make the product reusable across the product line - scenario 13, and conclude the results to utilize in the comparative analysis.

Table 11 contains all the identified scenarios for the sensitivity analysis, which describes the dynamic sensitivity factors/parameters to be modified in each scenario.

Note: Limitations of the analysis may include the exclusion of ECFSW's actual completion specifics, the applicability of the SCAT model to various flight software scenarios, and any informed assumptions made during the analysis.

For this study, we chose a minimal set of key parameters, as shown below in Table 10, for our sensitivity analysis. The choice of these parameters is driven by their effectiveness and specificity in assessing reusability. They act as sensitivity factors that define the input and output space of the model, which will be modified throughout different analysis iterations and scenarios.

Table 10. ECFSW SCAT/COCOMO Sensitivity Factors:

Category	Input Parameter
Size	New SLOC or Equivalent SLOC
	Initial Adapted SLOC
Reuse	Assessment and Assimilation (AA)
Reuse Software Understanding (SU)	SU – Structure
	SU – Application Clarity
	SU – Self-Descriptiveness
	Programmer Unfamiliarity (UNFM)
	Percent Design Modified
	Percent Code Modified
	Percent Integration Modified
Effort Multipliers	Developed for Reusability (RUSE)

In establishing different variations of the SCAT model, we make the assumptions in Appendix A which remain unchanged throughout the sensitivity analysis. It is important to keep in mind that these designations are very high-fidelity and educated assumptions since quantifying those efforts is extremely challenging as there are too many dynamic factors that could change day-to-day.

Comparative Analysis

In the previous section, we explored and compared the impact of various scenarios by changing some of the input parameters. This section analyzes and evaluates the results from the sensitivity analysis to identify similarities and differences between the 13 different scenarios to gain further insight into the strengths and weaknesses of the parameters being compared, as well as to make informed decisions about how they can be improved and optimized.

The activity accomplishes our objective by testing the comparison of the presumed alternative assumptions against the current/actual baseline model (Scenario 1) by comprehensively answering the following research questions:

- What are the better conditions to get bigger savings, and when do the assumptions undermine those savings?
- How do different inheritance choices (% pure reuse versus clone-and-own) change the effort estimates and how do their parameters compare? – Had we assumed higher inheritance for ECFSW, what would have that done to the cost?

Table 11. Sensitivity Scenarios & Alternatives

Scenario	Size		Reuse Parameters								Effort Multiplier
	New SLOC	Initial Adapted SLOC	Assessment & Assimilation	SU Structure	SU Application Clarity	SU Self-Descriptiveness	Programmer Unfamiliarity	% Design Modified	% Code Modified	% Integration Modified	RUSE
1	297,054	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
	365,539	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
	487,204	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
2	200,000	120,000	6	N	N	N	0.6	20	33	50	L
	262,300	167,000	6	N	N	N	0.6	20	33	50	L
	430,000	200,000	6	N	N	N	0.6	20	33	50	L
3	380,000	15,000	6	N	N	N	0.6	5	5	70	L
	465,500	24,500	6	N	N	N	0.6	5	5	70	L
	490,000	30,000	6	N	N	N	0.6	5	5	70	L
4	380,000	15,000	6	N	N	N	0.6	0	0	100	L
	465,500	24,500	6	N	N	N	0.6	0	0	100	L
	490,000	30,000	6	N	N	N	0.6	0	0	100	L
5	297,054	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
	480,000	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
	550,000	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
6	224,840	72,224	2	H	H	H	0.2	0	10	100	L
	250,595	114,944	2	H	H	H	0.2	0	10	100	L
	351,551	135,653	2	H	H	H	0.2	0	10	100	L
6.1	224,840	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
	250,595	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
	351,551	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
7	224,840	72,224	4	N	N	N	0.4	50	40	100	N
	250,595	114,944	4	N	N	N	0.4	50	40	100	N
	351,551	135,653	4	N	N	N	0.4	50	40	100	N
8	297,054	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	L
	365,539	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	L
	487,204	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	L
9	297,054	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	VH
	365,539	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	VH
	487,204	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	VH
10	148,527	148,527	2	N	N	N	0.4	50	50	70	L
	182,770	182,770	2	N	N	N	0.4	50	50	70	L
	243,602	243,602	2	N	N	N	0.4	50	50	70	L
11	89,116	207,938	2	N	N	N	0.4	25	25	100	L
	109,662	255,877	2	N	N	N	0.4	25	25	100	L
	146,161	341,043	2	N	N	N	0.4	25	25	100	L
12	72,224	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
	114,944	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
	135,653	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
13	72,224	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	VH
	114,944	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	VH
	135,653	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	VH

- ECFSW's FCPL cost absorption – How much did it cost EC to develop FCPL?
- FCPL cost isolation – How much effort has gone into developing FCPL, and how much more will it cost to make it highly reusable?
- FCPL cost – How much would it have cost to develop FCPL as a standalone product?
- FCPL development (dis)advantages – Are there any scenarios under which future projects will recoup the cost of developing FCPL through ECFSW from an institutional standpoint?
- ECFSW's FCPL cost penalty – What is it costing Clipper to make FCPL a reusable product and improve it?
- Going forward – Assuming ECFSW produces a stable or a mediocre version of FCPL, how does that impact future development when other projects inherit it?
- Planned versus actual inheritance cost savings – How much cost did inheritance actually save compared with the initial assumptions?
- ECFSW FCPL – How fast have FCPL & ECFSW modules been growing and how big will they end up getting?
- FCPL inheritability – How reusable is the current version of FCPL?

The comparative analysis concludes the results through the application of the comparison options in Table 12, expanded further in Table 15.

Table 12. Identified Comparison Options

Option	Comparison Description
1	Compare initial WM effort estimates and SLOC (at delivery) during project formulation to the current estimates of the project. Repeat the comparison for effort estimates at PDR and CDR phases, and projected completion
2	Compare initial WM effort estimates and SLOC (at delivery) during project formulation to the current SCAT effort estimates (baseline – scenario 1). Repeat the comparison for SCAT effort estimates at PDR and CDR phases, and projected completion (scenarios 2, 3, 4, 5)
3	Compare FCPL's current effort estimate by module size (SLOC) and WMs to the rest of ECFSW
4	Compare FCPL's SCAT WM effort estimates to the rest of ECFSW (Δ)
5	Compare module SLOC between software releases by determining the growth factor for FCPL and all ECFSW
6	Compare the current SCAT effort estimates (baseline – scenario 1) to the project's current WM estimates
7	Compare ECFSW with and without FCPL inheritance (pure reuse), SCAT estimates for scenario 6 against baseline – scenario 1

5. SENSITIVITY ANALYSIS

The sensitivity analysis described in the methodology section of the ECFSW requires the execution and assessment of the SCAT model for the 13 different scenarios shown in Table 11. The following sensitivity analysis will investigate 13 main scenarios based on the combination of constant/static parameters shown in Table 10 under different assumptions detailed in Table 11, observed in Tables 13 & 16, to identify the most impactful drivers and how they influence the total cost estimates and model outcomes.

An overview of what is to be expected in this section based on the detailed methodology in the previous section includes:

1. Establishing a baseline SCAT model for ECFSW.
2. Identifying key parameters for the sensitivity analysis.
3. Parameterizing the SCAT model for the specified scenarios, options, and alternatives under different assumptions or variations by the modification of a select set of parameters.
4. Recalculating SCAT effort estimates.

To estimate size parameters, the following steps are performed:

1. Decomposing the FSW considering heritage, functionality, and complexity.
2. Estimating size distribution parameters by counting logical lines of code or obtaining accurate estimates.
3. Deriving "Most Likely" based on analogous functions from the current software system.
4. Adjusting estimates for heritage and auto-generated code.
5. Selecting "Low" and "High" size estimates based on best/worst-case scenarios.

All size parameters use real data from Europa Clipper's FSW source lines of code acquired by using a source code counter tool called PySliC [15]. The rationale for the parametric assumptions in formulating the SCAT model instances is the following:

Scenario 1 – the basis of estimates provided in Table 13 is based on actual/current data collected from the project. The best-case scenario for the New SLOC is based on the projected size for a future major release R11.0.0 (not point-release), while the worst-case scenario is based on the underestimated size at project completion.

Scenarios 2, 3*, and* ***4*** – the basis of estimates is based on data collected from the project during the Initial, PDR, and CDR phases through SMART [14]. The best-case scenario for the New SLOC is based on the reported estimated size at project completion, while the Most Likely and Worst-Case scenarios are based on the difference between New SLOC or Equivalent SLOC & Initial Adapted SLOC**,** factoring in 33% and 5% reuse for the Initial Adapted SLOC respectively.

A comprehensive view of the scenario-based (non-linear) results is illustrated in Table 13.

Table 13. Sensitivity Analysis Results

Scenario	SCAT Model Output Summary			
	Module Unintegrated Effort (WM)	Mean Aggregate Eq. Size (KSLOC)	Mean Module Effort (WM)	MEAN TOTAL EFFORT (WM)
1	1,744.75563	425.42489	1,849.440967	1,849.440967
2	1,641.86978	421.5866136	1,740.381967	1,740.381967
3	1,980.240308	504.246138	2,099.054726	2,099.054726
4	1,976.731468	503.3924	2,095.335356	2,095.335356
5	2,027.233127	491.00998	2,148.867115	2,148.867115
6	1,355.118798	350.943455	1,436.425926	1,436.425926
6.1	1235.782561	305.98482	1309.929514	1309.929514
7	1,632.144994	399.1509606	1,730.073694	1,730.073694
8	1,657.517848	425.42489	1,756.968919	1,756.968919
9	2,006.468974	425.42489	2,126.857112	2,126.857112
10	1,398.261682	361.611471	1,482.157383	1,482.157383
11	1,186.047941	308.9860831	1,257.210818	1,257.210818
12	461.7085191	119.44377	489.4110302	489.4110302
13	530.9647969	119.44377	562.8226848	562.8226848

Scenario 5 – the basis of estimates is mostly based on projections. The best-case scenario for the New SLOC is based on the projected size for a future major release (R11.0.0) plus some informed delta, while Most Likely is based on the projected size for R11.0.0, and the worst-case scenario is based on the underestimated size at project completion.

Scenario 6 – the basis of estimates is based on the actual/current data collected from the project (similar to Scenario 1) with the consideration of the delta between FCPL and ECFSW modules SLOC as seen in Table 14 (had we assumed FCPL inheritance) – the difference between New SLOC or Equivalent SLOC & Initial Adapted SLOC and factoring in 100% "pure reuse" for the Initial Adapted SLOC.

Scenario 7 – the basis of estimates is the same as Scenario 6 except for factoring in a "Clone-and-Own" inheritance approach for the reuse parameters.

Scenarios 8 and 9 – the basis of estimates is the same as Scenario 1 except for factoring in Low and Very High RUSE effort respectively.

Scenarios 10 and 11 – the basis of estimates is the same as Scenario 1 except for factoring in 50% and 70% "Pure Reuse" effort for the Initial Adapted SLOC and reuse parameters, respectively.

Scenarios 12 and 13 – the basis of estimates is based on actual/current data collected from the project. The Most Likely for the New SLOC is based on the deducted actual/current FCPL SLOC as seen in Table 14. The best-case scenario is based on the projected size for a future major release (R11.0.0), while the worst-case scenario is based on the size at the half-point of FCPL's development (R6.0.0), which can be found in Appendix E.

Table 14. Europa Clipper FSW & FCPL Statistics

Europa Clipper FSW			
	ECFSW	FCPL (Δ)	Total
FSW Modules	60	46	106
Module Percentage (%)	56.60%	43.40%	100%
Actual WM	568.86625	357.56875	926.435
WM Percentage (%)	61.42%	38.59%	100%
Actual SLOC	250,595	114,944	365,539

6. COMPARATIVE ANALYSIS

This section compares the results from the previous sensitivity analysis in Table 13, as described in the methodology to highlight differences and significant changes throughout the various scenarios which will help us conclude sensitive patterns. This is accomplished by the following steps:

1. Analysis of the results – comparison of costs and parameter variations against the baseline and each other.
2. Identification of high sensitivity parameters – most significant impact on the estimated effort and cost.
3. Conclude sensitivity patterns.

By cross-referencing the current WM estimates and SLOC data acquired from the project as seen in Table 16, the resultant Figure 5 indicates that we have long surpassed the initial projected ECFSW size point.

A comprehensive view of the scenario-based (non-linear) results (rounded to the nearest whole number) is illustrated in Table 15.

Table 15. Comparative Analysis Results

Option	Comparison Description	Value (A)		Value (B)		Comparison (%)	
		WM	SLOC	WM	SLOC	WM	SLOC
1	Initial WM effort estimates and SLOC (at delivery) during project formulation vs. current estimates of the project	1,132	430,000	966	365,539	12.78%	16.20%
	PDR WM effort estimates and SLOC (at delivery) vs. current estimates of the project	1,695	493,000	966	365,539	51.97%	29.71%
	CDR WM effort estimates and SLOC (at delivery) vs. current estimates of the project	2,474	493,000	966	365,539	85.12%	29.71%
	Projected completion WM effort estimates and SLOC (at delivery) vs. current estimates of the project	1,286	297,054	966	365,539	25.40%	20.66%
2	Initial WM effort estimates and SLOC (at delivery) during project formulation vs. current SCAT effort estimates (baseline – scenario 1)	1,132	430,000	1,849	425,424	48.10%	1.07%
	PDR WM effort estimates and SLOC (at delivery) vs. current SCAT effort estimates (baseline – scenario 1)	1,695	493,000	1,849	425,424	8.69%	14.72%
	CDR WM effort estimates and SLOC (at delivery) vs. current SCAT effort estimates (baseline – scenario 1)	2,474	493,000	1,849	425,424	28.91%	14.72%
	Projected completion WM effort estimates and SLOC (at delivery) vs. current SCAT effort estimates (baseline – scenario 1)	1,286	297,054	1,849	425,424	35.93%	35.54%
3	ECFSW (Δ) current effort estimate (WM & SLOC) vs. FCPL	569	250,595	358	114,944	45.56 %	74.21%
4	ECFSW (Δ) SCAT effort estimate (WM & SLOC) vs. FCPL	1310	305,985	489	119,444	91.29%	87.67%
5.A	Module SLOC between software current release (R10.0.0) and projected future release (R11.0.0) via growth factor for all ECFSW	966	365,539	2,149	487,204	75.98%	28.51%
5.B	Module SLOC between software current release (R10.0.0) and projected future release (R11.0.0) via growth factor for FCPL	358	114,944	643	135,653	56.88%	16.51%
6	Current ECFSW SCAT effort estimates (baseline – scenario 1) vs. the project's current WM estimates	1,849	425,424	966	365,539	62.77%	15.14%
7	SCAT estimates ECFSW with 100% "pure reuse" FCPL inheritance (scenario 6) vs. baseline (scenario 1)	1,436	350,943	1,849	425,424	25.14%	19.15%
8	SCAT estimates ECFSW with "clone-and-own" FCPL inheritance (scenario 7) vs. baseline (scenario 1)	1,730	399,151	1,849	425,424	6.65%	6.37%
9	SCAT estimates ECFSW with 100% "pure reuse" FCPL inheritance (scenario 6) vs. "clone-and-own" FCPL inheritance (scenario 7)	1,436	350,943	1,730	399,151	18.58%	12.86%
10	SCAT estimates ECFSW with no RUSE effort (scenario 8) vs. baseline (scenario 1)	1,757	425,425	1,849	425,424	5.10%	0%
11	SCAT estimates ECFSW with very high RUSE effort (scenario 9) vs. baseline (scenario 1)	2,127	425,425	1,849	425,424	13.97%	0%
12	SCAT estimates ECFSW with no RUSE effort (scenario 8) vs. high RUSE effort (scenario 9)	1,757	425,425	2,127	425,425	19.03 %	0%
13	SCAT estimates ECFSW 50% pure reuse inheritance (scenario 10) vs. baseline (scenario 1)	1,482	361,611	1,849	425,424	22.03%	16.22%
14	SCAT estimates ECFSW 70% pure reuse inheritance (scenario 11) vs. baseline (scenario 1)	1,257	308,986	1,849	425,424	38.13%	31.72%
15	ECFSW 50% pure reuse (scenario 10) vs. 70% pure reuse inheritance (scenario 11)	1,482	361,611	1,257	308,986	16.43%	15.68%
16	FCPL-only SCAT estimates with nominal RUSE effort (scenario 12) vs. high RUSE effort (scenario 13)	489	119,444	563	119,444	14.07%	0%

The results in Table 15 suggest that the *lower* the comparison percentage, the *higher* the accuracy when comparing different estimates to each other such as SCAT WM to actual WM or initial WM seen in Options 1 and 2 and Option 6. However, when comparing different scenarios, a *larger* comparison percentage means *higher* sensitivity of the parameters that are being modified. Examples of this could be seen in Options 13 and 14 as a positive sensitivity pattern is observed when utilizing a pure reuse of 70% in the current baseline model which decreases the WM estimate from 1,849 to 1,257 and SLOC from 425,424 to 308,986.

Ideally, FCPL should be 80%-100% purely reusable. The comparison in Option 7 in Table 15, shows a significant decrease when purely reusing 100% of FCPL in the current baseline model which decreases the WM estimate from 1,849 to 1,436 and SLOC from 425,424 to 350,943, but not as significant as Option 8 when cloning and owning. Additionally, we can observe another relatively large impact in Option 9 when switching from a clone-and-own approach to pure reuse – a difference of 18.58% in WM estimates and 12.86% in SLOC. We also observe the impact of making a product reusable, in Option 12, which indicates that the WM estimates of the current baseline model when a Very High RUSE effort is assumed to increase from 1,757 to 2,127. Also, as seen in Option 11, when a Very High RUSE effort is assumed in the current baseline model that assumes Nominal RUSE, the WM estimates jump from 1,849 to 2,127.

Lastly, by measuring the rate of change between different software releases for ECFSW, we were able to realize a 1.1 growth rate factor in the software modules captured in Option 5.*A* with a 28.51% difference and 16.51% in 5.*B* in Table 15. Further detailed analysis can be found in Appendix B and E.

Table 16. ECFSW WM & SLOC Estimates [14]

Europa Clipper FSW					
	Initial (Estimate)			**Current /Actual**	**Completion (Est.)**
	Initial	**PDR**	**CDR**		
SLOC	430,000	493,000	493,000	365,539	297,054
WM Estimate	1,132	1,695	2,474	966	1,286
Assumed Inheritance	39%	5%	5%	< 5%	< 5%
% Code to be Modified	33%	5%	0%	N/A	N/A
SCAT/ COCOMO (WM)	1,740	2,099	2,095	1,849	2,149

7. CONCLUSION

NASA emphasizes the importance of flight software inheritance and reuse to improve the efficiency of flight software development processes, reduce costs, and maintain high standards of quality and reliability required for space missions [5]. This is evident from the sensitivity analysis performance results in Table 13 as the total estimated development effort decreased over higher inheritance and reuse rates. The comparative analysis conclusion favors FCPL inheritance both as “Pure Reuse” and “Clone and Own" scenarios with a reasonable estimate of integration effort versus no reuse at all. However, part of the analysis suggests that FCPL was not inherently designed for pure reuse, but it could be improved with a dedicated design effort to make it more modular and reusable, allowing it to be inherited more efficiently between flight projects.

In project formulation, accurate cost estimation drives optimistic and sustainable development budgets and is a contributing factor to mission proposal approval. A clear example of that is the Mars Sample Return mission budget at JPL which is currently constrained by NASA due to political pushback as a result of inconsistent cost management and fluctuating budget proposals. There is the repeated assumption in major proposals that future projects will use the Reference Bus and FCPL, also according to recent inheritance review reports, SRL – a sub-mission of MSR will most likely inherit from two different versions of FCPL, Europa Clipper and Psyche’s. Although not ideal, decision-making for reuse in cost estimation remains a challenging process. It requires a significant number of informed assumptions from prior missions with similar software and system architectures. Nonetheless, SRL would still benefit from inheriting the Psyche FSW hardware interface modules, primarily due to the close resemblance of its compute element to that of Psyche's.

According to our findings from the sensitivity and comparative analysis in Tables 13 & 15, the sensitivity factors match our presumed expectations. The better conditions for higher savings appear to be in a highly reusable environment. The inheritance and reuse data presented in this study highlight that continuing to use the “Clone-and-Own” model diminishes any optimistic chances of achieving relatively pure reuse. It also helped clarify the problem with this model in limiting reuse capabilities, that’s why utilizing a software product line like FCPL provides a break from the Clone and Own cycle. However, adopting a strictly “Pure Reuse” model might not be feasible after all, since Clone-and-Own might seem inevitable given each flight product is unique with a distinct set of goals and objectives, and a destination that will force its implementation to eventually diverge to fit the needs of the mission. This is partly due to the nature of work at JPL which is focused more on producing one-of-each versus scalable products, and projects’ core dependency on flight hardware.

The choice between “Clone-and-Own” and “Use-as-Is” in flight software inheritance should depend on the project's specific requirements, the level of customization needed, and

the trade-offs between flexibility, maintenance effort, and risk management. However, the chances of achieving pure reuse of FCPL on MSR/SRL remain very low, and that's because the FCPL implementation does not achieve all of its initial FSW Core partitioning goals but meets project needs and moves the effort closer to being able to accomplish it in future missions. Additionally, Europa Clipper did not make enough effort to make the product line reusable nor did Psyche as the projects were constrained and more concentrated on meeting their deliverables. However, a significant opportunity still exists to reduce potential future software costs even through cloning and owning. Considering the already incurred costs of developing FCPL on Clipper and Psyche, as well as its size, scale, and maturity, inheriting it for MSR will positively impact the overall cost. Lastly, the findings in this analysis can play an important role in identifying further software reuse trends and inhibiting factors to address issues related to cost overruns and make more informed decisions for future missions. Improving JPL's internal software management and oversight best practices by employing effective and efficient software reuse and higher inheritance rates will greatly contribute to obtaining a higher CMMI ranking during the next round of assessments which can significantly influence the center's ability to bid on more U.S. Federal Government contracts and acquire larger project funding that requires a certain level of process maturity.

APPENDICES

A. ECFSW SCAT/COCOMO STATIC PARAMETERS

Input Parameter	Values		
Size:			
Requirements Evolution and Volatility (REVL) Percent	8	10	15
Effort Multipliers:			
Required Software Reliability (RELY)	High	High	High
Test Database Size (DATA)	Low	Low	Low
Documentation Match to Lifecycle Needs (DOCU)	Nominal	Nominal	Nominal
Product Complexity (CPLX)			
CPLX - Control Operations	High	High	High
CPLX - Computational Operations	High	High	High
CPLX - Device Dependent Operations	High	High	High
CPLX - Data Management Operations	High	High	High
CPLX - User Interface Management Operations	N/A	N/A	N/A
Execution Time Constraint (TIME)	Very High	Very High	Very High
Main Storage Constraint (STOR)	Nominal	Nominal	Nominal
Platform Volatility (PVOL)	Nominal	Nominal	Nominal
Analyst Capability (ACAP)	Nominal	Nominal	Nominal
Application Experience (APEX)	High	High	High
Programmer Capability (PCAP)	High	High	High
Platform Experience (PLEX)	High	High	High
Language and Tool Experience (LTEX)	High	High	High
Personnel Continuity (PCON)	Nominal	Nominal	Nominal
Use of Software Tools (TOOL)	High	High	High
Required Development Schedule (SCED)	Nominal	Nominal	Nominal
Multisite Development (SITE)	Very Low	Very Low	Very Low
Scale Factors:			
Precedentedness (PREC)	High	High	High
Development Flexibility (FLEX)	Nominal	Nominal	Nominal
Architecture/Risk Resolution (RESL)	High	High	High
Team Cohesion (TEAM)	High	High	High
Process Maturity (PMAT)	High	High	High

B. SCAT MODEL ANALYSIS RESULTS

The SCAT Model Analysis Results detailed tables used in this paper are available in the GitHub repository titled "Data Sets and Tables for Evaluating Flight Software Effort Estimation and Reusability Approaches" under the filename "Appendix B. SCAT Model Analysis Results.xls" (https://github.com/SMikaelian/Data-Sets-and-Tables-for-Evaluating-Flight-Software-Effort-Estimation-and-Reusability-Approaches) [23].

C. Europa Clipper FSW Modules

The Europa Clipper FSW Modules table used in this paper is available in the GitHub repository titled "Data Sets and Tables for Evaluating Flight Software Effort Estimation and Reusability Approaches" under the filename "Appendix C. Europa Clipper FSW Modules.xls" (https://github.com/SMikaelian/Data-Sets-and-Tables-for-Evaluating-Flight-Software-Effort-Estimation-and-Reusability-Approaches) [23].

D. Europa Clipper FCPL FSW Modules

The Europa Clipper FCPL FSW Modules table used in this paper is available in the GitHub repository titled "Data Sets and Tables for Evaluating Flight Software Effort Estimation and Reusability Approaches" under the filename "Appendix D. Europa Clipper FCPL FSW Modules.pdf" (https://github.com/SMikaelian/Data-Sets-and-Tables-for-Evaluating-Flight-Software-Effort-Estimation-and-Reusability-Approaches) [23].

E. Europa Clipper FSW FCPL Modules Logical SLOC Statistics

The Europa Clipper FSW FCPL Modules Logical SLOC Statistics tables and data used in this paper are available in the GitHub repository titled "Data Sets and Tables for Evaluating Flight Software Effort Estimation and Reusability Approaches" under the filename "Appendix E. Europa Clipper FSW FCPL Modules Logical SLOC Statistics.xls" (https://github.com/SMikaelian/Data-Sets-and-Tables-for-Evaluating-Flight-Software-Effort-Estimation-and-Reusability-Approaches) [23].

F. SCAT Model Parameters

The SCAT Model Parameters table used in this paper is available in the GitHub repository titled "Data Sets and Tables for Evaluating Flight Software Effort Estimation and Reusability Approaches" under the filename "Appendix F. SCAT Model Parameters.pdf" (https://github.com/SMikaelian/Data-Sets-and-Tables-for-Evaluating-Flight-Software-Effort-Estimation-and-Reusability-Approaches) [23].

Acknowledgments

The research was conducted at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

The authors thank the Europa Clipper Project, Jet Propulsion Laboratory, California Institute of Technology, University of Southern California, and NASA Armstrong Flight Research Center for supporting this research. The authors also acknowledge the Europa Clipper Flight Software team, Jairus Hinh, Marcel Llopis, Klaus Havelund, Brent Morin, and Kenneth Cureton for their guidance and contributions to the work described in this paper.

REFERENCES

[1] Bocchino, Robert L. Jr., et al. "FPrime: An Open-Source Framework for Small-Scale Flight Software Systems." Jet Propulsion Laboratory, California Institute of Technology, 2018.

[2] Brown, James W. "JPL Reuse Program." Jet Propulsion Laboratory, California Institute of Technology.

[3] Cureton, Ken. "Concurrent Engineering Methods and Practices." Lecture #9, SAE 560, Astronautical Engineering Department, Systems Architecting & Engineering, University of Southern California (USC), 28 March 2022.

[4] "Europa Clipper Mission." NASA, Jet Propulsion Laboratory, California Institute of Technology, https://europa.nasa.gov/.

[5] Hirshorn, Steven R., Linda D. Voss, and Linda K. Bromley. NASA Systems Engineering Handbook. National Aeronautics and Space Administration, 17 Feb. 2017.

[6] Horvath, Gregory, et al. "Safety critical software architecture: a partioned software architecture for robotic spacecraft." Root, 2011, version V1, doi: 2014/42075, https://hdl.handle.net/2014/42075.

[7] Lum, Karen. "Software Cost Analysis Tool User Document". Version 1.2. Jet Propulsion Laboratory, California Institute of Technology, 18 Sept. 2006.

[8] Manglapus, Lloyd. "Core Flight Software Introduction and Status." Jet Propulsion Laboratory, California Institute of Technology, 30 Jan. 2018.

[9] "Mars Exploartion Program". NASA, Jet Propulsion Laboratory, n.d., https://science.nasa.gov/mission/mars-sample-return/.

[10] McComas, David. "Lessons from 30 Years of Flight Software." NASA Goddard Space Flight Center, Software Engineering Division, Flight Software Systems Branch, 22 Sept. 2015.

[11] Plakosh, Daniel, and James Anderson. "A Time and Space Partitioning Architecture for Safety-Critical Avionics." IEEE Transactions on Aerospace and Electronic Systems, vol. 41, no. 2, 2005, pp. 648-661.

[12] "Psyche Mission." Arizona State University, https://psyche.asu.edu.

[13] Stukes, Sherry, et al. "JPL Mission Software: State of Software Report 2018." JPL, September 2019.

[14] "Software Measures and Analysis Reporting Tool (SMART) repository." JPL, 2020.

[15] Taber, William. "Python Source Line Counter (PySLiC) User Guide", Release Version 1.0, JPL, May 22, 2019.

[16] Manglapus, Lloyd, and Tuszynski, Marek. "Avionics Subsystem CDR – Flight Software." Jet Propulsion Laboratory, California Institute of Technology, 5 Nov. 2020.

[17] Weiss, Kathryn Anne. "An Introduction to the JPL Flight Software Product Line." Jet Propulsion Laboratory, California Institute of Technology, 11 Dec. 2013.

[18] Wildermann, Charlie. "cFE/CFS." NASA Goddard Space Flight Center, Flight Software Systems Branch, 13 Nov. 2008.

[19] Stukes, Sherry. "JPL Awarded CMMI Level 3 Certification." Jet Propulsion Laboratory, California Institute of Technology, September 2022.

[20] M. Muszynski, E. Ferguson and S. Wissler, "The Evolution of Command and Sequencing at JPL: Origins and Flight Software Core Lineage," *2023 IEEE Aerospace Conference*, Big Sky, MT, USA, 2023, pp. 1-19, doi: 10.1109/AERO55745.2023.10115804.

[21] R. R. Moore and R. Sirohi, "Evolution and Analysis of Psyche's End-to-End Information System Architecture," 2022 IEEE Aerospace Conference (AERO), Big Sky, MT, USA, 2022, pp. 01-16, doi: 10.1109/AERO53065.2022.9843280.

[22] S. Khan et al., "Psyche: Innovations in Development of Planning and Sequencing Systems," 2024 IEEE Aerospace Conference, Big Sky, MT, USA, 2024, pp. 1-17, doi: 10.1109/AERO58975.2024.10521374.

[23] S. Mikaelian, "Data Sets and Tables for Evaluating Flight Software Effort Estimation and Reusability Approaches," *GitHub*, 2025. [Online]. Available: https://github.com/SMikaelian/Data-Sets-and-Tables-for-Evaluating-Flight-Software-Effort-Estimation-and-Reusability-Approaches. [Accessed: 17-Jan-2025].

BIOGRAPHY

Sarkis Serge Mikaelian *received a B.S. in Computer Science from California State University, Northridge, and a Master's in Systems Architecting & Engineering from the University of Southern California where he was awarded the Systems Architect & Engineer of the Year recognition. He has worked at NASA Armstrong Flight Research Center as a Flight Software Engineer in the Resilient Autonomy and X-57 programs. He spent a year working at NASA Langley Research Center as an Artificial Intelligence researcher for Intelligent Contingency Management. He currently works at JPL as a Flight Software Integration and Test Engineer, and Avionics Operations engineer in the Advanced Flight Software group for Europa Clipper. During his time at JPL, Sarkis held additional roles as a Flight Systems Engineer, developing FSW requirements for the Mars Sample Return mission, and verifying & validating the CADRE autonomous lunar surface exploration rovers.*

Marek Tuszynski *earned a B.S. in Mathematics and Computer Science as well as a Master's in Applied Mathematics from California State University Northridge. Marek has been with JPL for more than 36 years working in project, line, and institutional roles. He is currently engaged as Product Delivery Lead for Europa Clipper Flight Software (FSW) with an additional assignment as JPL's Software Process Owner, or Technical Authority for the Software Development Requirements. Marek has also held positions as Section Manager for Flight Software and Avionics Systems and Product Delivery manager on Mars Science Laboratory FSW, Technical Group Supervisor of the Guidance and Control FSW Testing group, Project Element Manager on the Space Interferometry Mission, and Cognizant Engineer for the Cassini Spacecraft Attitude Control FSW. Marek was awarded Principal designation in recognition of his sustained outstanding individual contribution.*

Lloyd Manglapus *received his B.S. and M.S. degrees in Computer Science from the University of Southern California. He has been developing flight software at JPL for more than 24 years. He is currently managing the technical development of the Europa Clipper spacecraft flight software. Prior to Europa Clipper, he was a co-architect of the JPL Flight Software Core Product Line and supervised its development. He also developed flight software for various spacecraft including Mars Science Laboratory. He started his career at JPL by developing instrument flight software for the Tropospheric Emissions Spectrometer.*