REDUCING FLIGHT SOFTWARE DEVELOPMENT COST RISK: ANALYSIS AND RECOMMENDATIONS

Jairus M. Hihn, Ph.D.
Hamid Habib-agahi, Ph.D.

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, Ca. 91109*

ABSTRACT

This paper documents a set of recommendations for reducing flight software cost risk as derived from a detailed analysis of the underlying causes of the observed cost growth. The results are based on a detailed study of eight deep space missions that experienced significant cost growth between 1995 to 1999. The average software development cost growth for these missions was 51%. The approach used integrated information gathering and synthesis techniques from multiple methodologies, protocol analysis, affinity analysis, focus groups and multi-voting. It is the conclusion of this study that there will be an increase in cost growth, schedule slips and software failures on future missions unless the visibility and importance of software at the project level is increased. Furthermore, there must be a shift in focus from a hardware oriented system design to the development of hardware/software integrated system and subsystem architectures. Relative to the way that NASA has traditionally built its spacecraft the most significant recommendation in support of these objectives is the creation at a project level of software system management position with oversight and authority over ground and flight software

1.0 INTRODUCTION

The Jet Propulsion Laboratory (JPL) is a US Government Federally-Funded Research and Development Center that is run by the California Institute of Technology for the National Aeronautics and Space Administration (NASA)1 JPL's primary role is to conduct unmanned, robotic space exploration missions throughout our solar system. JPL has a long record of successful deep space missions from Explorer to Voyager, to Mars Pathfinder. JPL's experience and success as with the rest of the aerospace industry is built upon our hardware and system level expertise. The majority of JPL's, as well as other aerospace organizations, current managers and system engineers have made their reputations on these hardware intensive spacecraft. Only recently has software become more important in its contribution to spacecraft risk, integration and labor cost, therefor more emphasis on software development management and planning is required.

2.0 BACKGROUND

JPL, along with the rest of NASA and industry, is actively engaging the "faster, better, cheaper" philosophy. The first "faster, better, cheaper" mission was Mars Pathfinder. It was 70% less than the average cost of JPL's major space missions from 1964 to 1994. In addition, the frequency of missions launched has dramatically increased, from the historical rate of about one mission every two years to multiple missions launched every year. This shift has required major changes in the way JPL does business and is creating some institutional strain as the organization finds ways to adapt.

In response, JPL is undergoing a radical redefining of its development processes to provide future missions the means to achieve the goals of "faster, better, cheaper". The focus of these activities until very recently has been primarily on improving the development process and tools for hardware employing concurrent engineering and information technology. Also underway is what appears to be a very successful shift towards incorporating off the shelf hardware for many mission components. However, during this same time of seeking new ways of doing business, spacecraft have become more software intensive.

This paper reports the final results of a detailed study of eight missions that experienced significant cost growth between 1995 to 1999[1]. A summary of the high level causes and proposed flight software cost risk model is reported in Hihn and Habib-agahi[2]. In this paper the underlying causes of the cost growth are identified. Based on a detailed analysis of the software cost growth causes a set of policy recommendations are proposed.

3.0 METHODOLOGY

3.1 Sample Definition

In order to limit study cost and time, it was decided not to make an exhaustive or stratified survey of all flight software under JPL management.

Missions were included in the study based on the following criteria:

- Cost growth had to exceed 20% of plan at PDR in the last 3 years, except one mission had to be included that stayed within its budget
- At least one ground support software development task had to be included
- A mixture of in-house and subcontracted missions had to be included.
- Participants were chosen based on having worked extensively on the selected missions

3.2 Data Collection Methodology

The data collection methodology was relatively complex consisting of multiple steps:

1) Interviews
   a) Unstructured Interview based on Protocol Analysis to obtain self reports of what happened on specific missions
   b) Follow up Structured Interview to verify how self reports had been categorized and to identify missing information
2) Workshop: Cost Growth Causes. A focus group to brainstorm underlying causes of software cost growth based on interim findings from the initial interviews
3) Multi-voting to verify top cost risk categories
4) Workshop: Policy Recommendations. A focus group to review and finalize JPL strategic software policy recommendations

3.2.1 Interviews

The two interview sessions consisted of approximately 60-90 minutes each. The interviews primarily focused on a single selected mission. Two to three persons conducted both interviews. One interviewer functioned as the main scribe and interviewer, the others as backup to reduce the likelihood that information was lost or a potentially important point was missed. The approach used is a modification of the Protocol Analysis methodology[2]. It translates self-reports into ordinal data by grouping descriptive information into categories.

The first interview was relatively unstructured and consisted of only four basic questions. The open ended interview questions were designed to elicit information concerning what was working and not working within the selected mission/project. Detailed notes were taken from each interview, which were transcribed and used as the information source from which data could be derived. The interviewees typically responded to the questions by describing specific events or behaviors that supported their response and illustrated their issues or concerns.

The questions for the first interview consisted of:

1. Identify mission, mission objectives, and interviewees mission role
2. Describe the main causes of the software development cost growth experienced on your project
3. (Identify the top three software development cost risks based on your experience
4. Describe what you will do differently, to reduce the software development cost risks you identified

After the interviews were completed and transcribed, the responses were reviewed and systematically grouped into common themes, which resulted in the identification of 7 major cost growth risk areas or categories. After developing a first draft set of causes and potential recommendations, a second interview with the same group was initiated in order to review how their information had been mapped into the identified cost growth categories. The respondents could modify the information, add new information, delete information and even add new categories. In addition, the participants were asked to provide a subjective estimate of the overall cost growth as a percentage of the budget/plan at the time of the Preliminary Design Review (PDR) and the percentage contribution of each cost growth category. The tables were updated based on the information provided by the participants. The analysis was performed based on this information. An example of a cost risk table from an actual interview is displayed in Figure 1.

3.2.2 Workshop: Cost Growth Causes

A four-hour workshop was scheduled to which all of the respondents were invited; in addition several members of the Center for Space Missions Information and Software Systems (CSMISS) team were also invited to join in. The objective of the workshop was to verify the results of the information gathered in the previous interviews and to identify a more detailed list of causes for cost growth. To help the participants generate a more detailed list of cost growth sources the participants were repeatedly asked, why a cause existed. The question of "why" continued to be answered until no new explanations were forthcoming and then the group moved on to the next cause that had been previously identified (See example below). Participants also categorized the causes by whether they primarily were due to a problem with JPL's processes, Tools & Methods, people/teams, or culture.

Risk Area	Percentage contribution to cost growth	Participant Statements Grouped by Risk Area
Experience & Teaming	10%	• System engineers had extensive HW experience but had limited SW experience • Managers were HW oriented and had insufficient understanding of SW
Planning, Estimation & Control	50%	• Software development cost was underestimated because (1) had assumed too much inheritance and (2) had underscoped effort partly because had not accounted for code growth.
Requirements & Design	10%	• Did not have sufficient traceability between system and SW requirements and also had no process to feed back information as learned more about SW requirements.
Testing	15%	• Insufficient testbeds • Simulators were not ready until late in lifecycle, which delayed testing • Testbeds lacked capability (sufficient functionality)
Software Inheritance	included in planning	• Assumed software inheritance of 30% from a previous mission but ended up inheriting less than 5%.
Tools & Methods	10%	• Missed some test results because had bad analysis tools
Staffing	5%	• Persistently under staffed.
TOTAL Cost Growth	50%	• Project completed with 50% cost growth in software development costs

Figure 1: Example Cost Risk Table Used in Interviews

An example sequence was:

Software Cost Growth Cause (from interviews):
There is a lack of software experience in the project office.

Why? Many managers and system engineers have limited software experience

Why? (1) Most managers and system engineers grew up in a hardware world.
(2) Managers and system engineers do not view software engineers as broad enough.

Why? (1) JPL has a very mature and dominant hardware culture
(2) The software culture is still in the process of maturing

3.2.3 Multi-Voting

The objective of this part of the study was to identify a set of key policy recommendations especially from a strategic perspective that would address the largest percentage of the software risk areas. Strategic recommendations are actions that require institutional support. Tactical recommendations are actions that missions can implement without direct institutional support. The approach used was to send out an Excel worksheet that contained two voting forms to the participants, see Figures 2 and 3.

The first voting form, see Figure 2, requested that the participants allocate 11 votes in response to the following questions, "Which risk areas do you think are the most important for JPL to address?" (Strategic) and "Which risk areas do you think are the most important for project managers to consider?" (Tactical). The objective was to verify that the top risk areas that had been identified in the earlier parts of the study had not changed after the extensive discussions. The identification of the top risk areas is used to narrow the search for the key policy recommendations. Participants were asked to allocate n+1 votes over n recommendations and then to specify whether they perceived the difficulty of implementing the recommendation as hard, medium or low. There was one form for each of the eleven risk areas.

The second voting form grouped the 46 recommendations by the risk areas with which they had the highest affinity. The participants were asked to allocate N+1 votes over each grouping of the recommendations, where N is the number of recommendations in the set. In addition, they were asked to rate the ease of implementing the recommendation as Hard, Moderate, or Easy. See Figure 3 for a sample of the voting form used. These results are then used to actually identify the top recommendations in the top risk areas. It is from this reduced set of recommendations that the policy statements are derived.

3.2.4 Workshop: Policy Recommendations

A second workshop was held jointly with the Center for Space Missions Information and Software Systems (CSMISS). The objectives of the workshop were to (1) Review and verify the proposed policy recommendations; (2) Verify that all relevant recommendations from this study were included in the CSMISS software principles as appropriate.

Directions: You have 11 votes for each question, which you may allocate in any manner that you wish. For example, you may place all the votes on one Risk Area, spread them equally, or any combination in between as long the total number of votes adds to 11.		
	Which risk areas do you think are the most important for JPL to address? Enter number of votes below.	Which risk areas do you think are the most important for project managers to consider? Enter number of votes below.
Risk Area	**Q1: Strategic**	**Q2: Tactical**
1. Experience		
2. Teaming		
3. Control		
4. Planning and Estimation		
5. Software Reuse		
6. Architecture		
7. Software Volatility		
8. Staffing		
9. Tools & Methods		
10. Test Beds and Test Tools		
11. Testing Practices		
TOTAL VOTES		

Figure 2: Multi-Voting Form Used to identify Top Risk Areas

4.0 DATA SUMMARY

A total of 11 managers and engineers provided information for this study. They held a variety of positions from Technical (Cognizant Engineer) to Project Manager. The study included software from 8 missions out of 24 that were currently either in development or operations. There are six flight and two ground systems with some having completed implementation and some still under development; three were subcontracted; and only one of the missions included in the study has not exhibited any cost growth.

This shows that the average cost increase for projects that experience cost growth, is 51% with a range of 25% to 71%, excluding the highest and lowest observations. Of course, one should not conclude that all flight software developments would exceed their PDR plan by 50% based on the results of this study. The sample size is small and the flight software tasks were preselected based on the condition that they had more than 20% cost growth (except for one) over the plan at PDR.

	You have 4 votes to allocate over the following recommendations. Please allocate them to indicate your perception as to the importance of implementing the recommendation	Please rate your perception of the degree of difficulty in the implementation of each recommendation. Use 3 for High, 2 for Medium 1 for Low.
Recommendation	**Importance**	**Ease of Implementation**
Project office needs to have some SW expertise		
Everyone should have some mission level training to provide end-to-end understanding of the system		
Software team needs to understand the system		
Need system engineers who understand SW		

Figure 3: -Multi-Voting Form Used to Identify Top Recommendations

5.0 FLIGHT SOFTWARE COST GROWTH CAUSES

5.1 Flight Software Cost Growth Sources and Impact

Based on a categorical analysis of the data a number of key risk areas were identified. These areas are: Experience & Teaming, Planning, Requirements and Design, Testing, Software Inheritance, Staffing, and Tools & Methods.

Figure 4 contains a summary of four different perspectives from which to rank the importance of each risk category.. Here it can be clearly seen that from each of the perspectives (frequency of occurrence, estimated impact, strategic importance, and tactical importance) that the top three sources of flight software cost growth are: Planning, Requirements & Design, and Experience & Teaming, which represents approximately 70% of the potential sources of cost growth. See Hihn and Habib-agahi[1] for a more extensive discussion of the high level sources of flight software cost growth. The next step is to more fully explore the underlying causes associated with these risk areas.

Risk Area	Frequency of Occurrence (% of Projects)	Estimated Contribution To Cost Growth	Which risk areas do you think are the most important for JPL to address?	Which risk areas do you think are the most important for project managers to consider?
Planning (incl. Control)	71%	35%	28%	28%
Requirements & Design (incl. Architecture & SW volatility)	57%	25%	15%	13%
Experience & Teaming	71%	10%	20%	25%
Testing	71%	15%	14%	10%
Staffing	71%	10%	7%	10%
Software Inheritance	57%	Incl. In Planning	11%	3%
Tools/ Methods	86%	5%	5%	6%

Figure 4: Summary of Software Cost Growth Sources by Importance

The results of the multi-voting process were used to identify which of the top risk areas could be mitigated by the projects independently and which required JPL level support. Several changes in the risk areas were made for this step as a result of the discussions in the Policy Recommendations Workshop, Experience and Teaming were separated, Requirements & Design was converted to System and Software Architecture. The top three risk areas received over 50% of the votes. The results show that the most important risk areas to address from either a strategic or tactical perspective are almost identical. The differences are that software inheritance is seen as primarily requiring institutional support while staffing and control problems can be dealt with by the projects.

5.2 Detailed Identification of Software Cost Growth Causes

Based on the results and follow-on discussions with the participants it was decided to focus on the top five strategic risk areas:

1. Planning
2. Requirements & Design
3. Experience & Teaming
4. Testing
5. Software Inheritance

Figure 5 provides a summary of the results of the workshop discussion on the detailed sources of flight software cost growth for the top five risk areas. The identified causes are primarily "process" related, then "people & team" and last are "tool" related causes. Sixteen Process related causes were identified, which are associated with 4 out of 5 risk areas. The process related causes are diverse, however, a common theme that runs through many of them is that software is not receiving sufficient visibility at the system level or early enough in the lifecycle. More specifically, the causes identified were related to: starting planning and requirements & design activities too late in the lifecycle; insufficient review of the software plans and design in the early phases of the lifecycle, especially with respect to software inheritance; and lack of a fully integrated hardware/software system architecture. Thirteen People/Team related causes were identified, which are associated with all 5 of the risk areas. The People/Team causes expand upon the issues already identified in the initial interviews, lack of sufficient software experience on the part of management and system engineers and also a lack of end-to-end mission experience on the part of the software engineers, which is further compounded by communication breakdowns between the hardware and software engineers. This analysis does reveal that these problems with experience and teaming play a significant role in reducing software cost risk in all categories. Nine Tool & Methods related causes were identified, which are associated with 3 risk areas. The most critical items relate to the lack of availability of testbeds and also incorrect assumptions with respect to the ease of inheriting software and incorporating COTS.

Risk Area	Cost Growth Sources	Cost Growth Causes		
		Process	People/Teams	Tools & Methods
Planning	• Poor planning and estimation practices • Insufficient reserves for SW	• No generally accepted planning process for software development.; planning is largely dependent on the individual engineer (preparing the plan) • Uniqueness of software not captured in initial stages (functional to deliverable) • SW requirements and design are more volatile & solidify later than hardware in the life cycle. • Don't know how to freeze software requirements the same way we know how to freeze hardware requirements	• SW team not included in early stages of planning • SW not recognized in initial planning	• Poor and constantly changing assumptions and cost estimation methods • Lack of software planning tools • Lack of SW cost metrics.
Requirements & Design	• Lack of good architecture and system partitioning • Systems decisions made without accounting for impact on software	• Subsystem view of spacecraft -- not viewed as important to have a top-level architecture early in the project. • Software design is traditionally done at the subsystem level (based on hardware perspective) • Architectural issues are not sufficiently worked out in Phase A/B • Concurrent development can lead to interface problems due to lack of communication between teams especially when there is schedule compression.	• No awareness or recognition even at the mission & system level that software needs to be addressed. • Don't view architecture as a software intensive process	
Experience & Teaming	• Insufficient software experience among managers and system engineers • Poor teaming between HW/ SW and systems/SW team		• Management and system engineers have limited SW experience • Engineers grew up in a hardware intensive world. • Managers and system engineers do not view software engineers as broad enough. • Lack of software-system engineers • Software culture is underdeveloped at the present	
Testing	• Testbeds; too few, too late, not validated, insufficient capability • Lack of early test planning; lack of functionality,	• Lack of sufficient funding. • Testbeds not listed in WBS; not accountable. • Lack of sufficient schedule or recognition of the importance of testing. • "Big Bang" style testing waits until end to test. • Test documents not in place until late in life cycle	• Lack of education & appreciation of value for testbeds. • Test team not in place until late in life cycle • Integration and SW teams not available to support ATLO	• Dependence on hardware testbeds. • Lack of tools and under utilization of existing tools • Lack of controlled tests and test data
Software Inheritance	• Inherited code did not behave as advertised, was poorly documented, and required more modification than expected	• Lack of software inheritance review process. • Inheritance not distinguished between reusable code and code that has not been designed for that purpose. Inheritance (typically) only reuses the design. • No incentives for projects to develop fully reusable code.	• Many projects fail to bring onboard the original developers when they attempt to inherit software	• Too many advantages of inheritances assumed, esp. cost savings • Cost models don't properly account for COTS, sw inheritance and modification.. • Too often assumed that COTS costs are free

Figure 5: Top Five Risk Areas: The Causes Flight Software Cost Growth

Risk Area	Summary of Reported Recommendations
Experience & Teaming	•Need project managers & system engineers who understand SW •System engineers need to understand that SW provides the system level interfaces •Project office needs to have some SW expertise •Need to build a team that can work together and communicate (includes cross hardware/software) •PMs need to be able to identify staffing problems early
Planning	•Need a focused end point with clear success criteria •Need better tailored risk management with contingency plans • Need a plan you can track and hang your hat on based on a complete lifecycle •SW must have an early presence even in pre-Phase A and be part of an integrated plan • Allocate larger reserves to SW •Require that a clear understanding of SW be included as part of NAR approval • Need more detailed planning and tracking of SW similar to HW • When putting together a plan get inputs from everyone and negotiate. Add schedule slack but make sure all manager's know they are accountable • Need to change rules of thumb. E.g., SW development vs. test used to be 50/50 now appears to be 15/85
Requirements & Design	• Must have a development process that deals with evolving reqs & assumes things will break. Early and extensive prototyping Incremental deliveries & evolving documents Isolate interfaces • Identify standardized SW functions and put in HW. •Need good architecture to define demarcation between HW and SW. •Do not look at SW as separate but see as an integrated design • Get a baseline and CM in place so can carefully manage prioritized requirements
Testing	• Need to have many and varied SW test environments • Need to have a dedicated integration and a dedicated test team whose job it is to break the SW • Testbeds and simulators need to be made a major product deliverable that is completed early in lifecycle
Software Inheritance	•Need a software inheritance review •For successful software inheritance, developers need to come with the software
Tools etc.	• Make sure target and development systems are the same • Use design tools with proven record • Get methodology and process in place before purchasing tools • Need good test analysis tools
Staffing	• We need to go outside to get more expertise • Software team needs to understand the system •Plan to over staff SW engineers to deal with turnover •Need a mechanism to hire more SW people without elaborate hiring procedures • Everyone should have some mission level training to provide end-to-end understanding of the system

Figure 6: Summary of Reported Recommendations by Risk Area

6.0 FLIGHT SOFTWARE COST RISK RECOMMENDATIONS

Similar to the methodology used to identify the causes of software cost growth, a multi-step approach was used:

1. Identified a set of initial recommendations based on individual participant's responses,
2. Mapped recommendations into detailed risk causes (list produced at the first workshop)
3. Select top five risk areas by multi-voting (see results in section 4.1)
4. Identified key recommendations associated with each top risk area by multi-voting
5. Derived a set of policies to implement the key recommendations for each risk area (Verified policies at second workshop).

6.1 Initial Recommendations based on Lessons Learned

During the initial interviews, participants were asked to provide recommendations as to what needs to be done to reduce the potential for flight software cost growth. Figure 6 contains a summary of the recommendations that were generated during the initial interview sessions. The recommendations that came up repeatedly (more than 50% of the time) were:

- Project managers & system engineers must have a better understanding of software
- More detailed planning and tracking of SW similar to HW is required
- SW must have an early presence even in pre-Phase A and be part of an integrated plan
- The software development process must deal with evolving requirements & assume that the unexpected will happen. This requires the use of:
  Early and extensive prototyping;
  Incremental deliveries with evolving documents;

Isolation of interfaces through layered and modular design

It can be clearly seen that many of the recommendations directly or indirectly relate to dealing with software requirements volatility, partial and incomplete information in the early phases of a mission, as well as basic communication issues. The communication issues are arising because the vast majority of project managers and senior system engineers came to maturity in a hardware oriented world, while the software engineers frequently have a limited understanding of the mission and spacecraft as a system. Traditionally, there are also fundamental differences in how system and software engineers develop their designs and the way in which interfaces are described. This has caused miscommunications on a number of JPL's major missions. Two different approaches are proposed for dealing with rapidly changing software requirements that is a natural part of an R&D project. One approach is to try to obtain more detailed software requirements information and complete more detailed planning earlier in the lifecycle. The other is the use of a rapid development approach that is based on the premise that detailed software requirements do not exist in the early phases of the mission lifecycle and that overly detailed plans are counter productive.

6.2 Key Recommendations Identified by Multi-Voting

A multi-voting approach was used to identify the top recommendations corresponding to each risk area. All participants were requested to identify their top recommendations from the original set collected during the interviews. Additional recommendations were also added as a result of the analysis of the detailed cost growth causes.

The list of the sixteen top recommendations is displayed in Figure 7. Fourteen of the recommendations received at least one or more votes from each participant, within the top five risk areas, during the multi-voting exercise. Two additional recommendations were identified based on using a weighted voting algorithm derived by combining the number of votes received by each key risk area with the normalized number of votes received by each areas respective recommendations. These are recommendations 9 and 10 in Figure 7. While the objective is to identify a number of key policy recommendations that have been endorsed by the participants, the complete listing of recommendations should be reviewed, as all but a small number of recommendations received at least one vote per participant.

The key recommendations have been grouped into three categories similar to those in Figure 5 in section 5.2. The categories are Process, People/Teams, and Tools/Methods. The process recommendations focus on earlier development of key design, software, test, and management products and earlier and more frequent reviews of software products and plans. The People/Team recommendations focus on the need to have more cross cutting experience in our project teams, and in bringing software and test engineers on-board earlier in the lifecycle. Only one recommendation in the Tools/Methods category received enough votes, which related to the need for institutional support to greatly increase the inheritance of software between projects.

6.3 Policy Recommendations

The final step was to translate the key recommendations into specific JPL software policies that can be implemented by managers of future missions. While some of the recommendations map very simply to a potential JPL policy, many of the recommendations are a combination of what to do and how to do things differently. They are also at different levels of granularity. The list of key policy recommendations, shown below, was finalized during the second workshop.

The policy recommendations are grouped by whether they primarily require an organizational change, product change or process change. The main theme of these recommendations is that the visibility and focus on software must be raised to a much higher level. This issue arose time and time again in all of the interviews and workshops and requires significant changes in how we organize a project and design our missions. These policy recommendations are consistent with the dramatic changes we have seen in the nature of spacecraft as software has come to play an increasingly important role in spacecraft integration and automation.

(A) Recommended JPL Organizational Policy

1. Require all projects have a software system manager with budget authority and responsibility over flight and ground SW and reports directly to the project manager. (The same as the spacecraft and instrument managers.) Among others the software system managers responsibilities include:

- Preparation of software cost estimates, plan, and budget
- Development of software architecture by PDR
- Ensure consistency of software architecture and the system architecture
- Ensure that software is considered in all design trades
- Supporting subsystem managers in planning, development, integration, test, operations, and maintenance.
- Coordination of operations, flight software, and ground software.
- Determine how software will be managed within the project and integrated within the overall project implementation structure.

(B) Recommended JPL Product Policies

2. Require the development of a system architecture supported by a software architecture that clearly documents an integrated hardware and software design prior to PDR.
3. Require the development of a management plan that addresses software including a risk management plan with reserve and contingency allocations based on estimated risk prior to PDR.
4. Require the development of a test strategy and plan prior to PDR.

(C) Recommended JPL Process Policies

5. Require a Software Inheritance Review similar to the Hardware Inheritance Review (when appropriate) prior to PDR and CDR.
6. Require that software requirements and architecture be reviewed at the NAR.
7. Require that the software architectural designs be reviewed at PDR and updated at CDR.
8. Require Risk Management Plan be reviewed at PDR and updated at CDR.
9. Require Test Plans and status be reviewed at PDR and updated at CDR.

7.0 CONCLUSIONS

A number of JPL managed missions have experienced cost growth with respect to the flight software portion of the mission. This has occurred for both in-house and subcontracted missions. The results from this study indicate that the sources of flight software cost growth can be categorized into a small number of basic risk areas with three accounting for 75% of the cost growth, Planning, Requirements & Design, and Testing. Two other key risk areas relate to Software Inheritance and Experience & Teaming. The study results also indicate that given the current software development environment and approach, that software development cost reserves greater than 30% are likely to be required.

While this study is based on a small sample, managers should consider paying more up front attention to these risk areas during the planning phases of a mission. A major implication of this study is that missions need to increase the visibility of software at the system and project level. Key recommendations consistent with the study results are:

- Create a software position with budgetary authority at the project office level,
- Develop an integrated HW/SW system architecture supported by an integrated software architecture
- Generate software design, plan and test documents early with continuous review throughout the project development lifecycle phases.
- Review rigorously and early any planned software inheritance and COTS usage, similar to the existing hardware inheritance review.

BIBLIOGRAPHY

1. Hihn, J. and Habib-agahi, H. Flight SW Cost Growth, JPL D-18660, Internal Document, January, 2000.
2. Hihn, J. and Habib-agahi, H. Identification and Measurement of the Sources of Flight Software Cost Growth, Proceedings of the 22nd Annual Conference of the International Society of Parametric Analysts (ISPA), 8-10 May, 2000, Noordwijk, Netherlands
3. Simon, H. and Ericson, K., Protocol Analysis; Verbal Reports as Data, MIT Press, 1993

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology under a contract with the National Aeronautics and Space Administration.

Risk Area	Cost Growth Sources	Recommendations		
		Process	People/Teams	Tools/Methods
Planning, Estimation & Control	• Poor planning and estimation practices • Insufficient reserves for SW	1. Need a focused end point with clear success criteria 2. Need better tailored risk management plan with appropriate contingencies 3. Allocate larger percentage reserves to software		
Requirements & Design	• Lack of good architecture and system partitioning • Systems decisions made without accounting for impact on software	4. Require that a clear understanding of SW be included as part of NAR approval 5. Need good architecture to define demarcation between HW and SW	6. System Engineers need to understand that the software provides the system level interfaces 7. Do not look at SW as separate item but see as part of an integrated system design	
Experience & Teaming	• Insufficient software experience among Managers and system engineers. • Poor teaming between HW/ SW and systems/SW team		8. Project office needs to have some SW expertise 9. SW team needs to understand system[1] 10. Everyone should have some mission level training to provide end-to-end understanding of the system [1]	
Testing	• Testbeds; too few, too late, not validated, lacked capability • Lack of early test planning; lack of functionality,	11. Testbeds and simulators need to be made a major product deliverable that is completed early in lifecycle	12. Need to have a dedicated integration team and a dedicated test team whose job is it to break the software 13. Require a test engineer be a member of the early planning team and reviews.	
Software Inheritance	• Inherited code did not behave as advertised, was poorly documented, and required more modification than expected	14. Need a software inheritance review	15. For Inheritance people need to come with the software	16. To increase the amount of Inheritance between projects, need to create infrastructure to provide incentives to develop reusable code and to maintain it.

Figure 7: Recommendations in Top Risk Areas Receiving 10 or More Votes

1. Recommendation identified based on combining the key risk area and recommendations multi-voting results to provide a ranking of recommendations that crossed the risk areas.