Assuring Software Cost Estimates: Is it an Oxymoron?

Jairus Hihn[1]
Jet Propulsion Laboratory
California Institute of Technology

Grant Tregre
NASA
Goddard Space Flight Center

Abstract[1]

The software industry repeatedly observes cost growth of well over 100% even after decades of cost estimation research and well-known best practices, so "What's the problem?" In this paper we will provide an overview of the current state of software cost estimation best practice. We then explore whether applying some of the methods used in software assurance might improve the quality of software cost estimates. This paper especially focuses on issues associated with model calibration, estimate review, and the development and documentation of estimates as part of an integrated plan.

1. Introduction

The problems associated with inaccurate software cost estimates are well documented. Early lifecycle effort estimates can be inaccurate by up to 400% [1, p310]. In a study of NASA software development projects conducted in the late nineties, the most frequently identified cause (71%) of cost overrun with the largest impact (35% contribution to observed cost growth) was basic failures in planning, estimation & control [2]. In the worst case, over-running projects are canceled, resulting in the waste of development efforts. For example, in 2003, NASA canceled the CLCS (Check Out Launch Control System) project after spending hundreds of millions of dollars on software development. The project was canceled after the initial estimate of $206 million was increased to between $488 million and $533 million. Upon cancelation, approximately 400 developers lost their jobs [3].

These problems continue even after years of research on cost estimation techniques, the development of sophisticated cost models [1], the creation of cost related professional societies and certification programs for professional cost estimators [4]. Many companies have defined requirements, cost processes and standardized templates to address cost estimation problems. In spite of these contributions to defining Software Estimation Best Practices (BP), the cost estimation community continues to struggle with producing good cost estimates. This seems to be even more of an issue in the aerospace and high tech industries. One contributor to such inaccuracies is that software engineers and managers continue to perform bottom up estimates with little or no data to support their assumptions and little or no consideration for risk and uncertainty [5]. Software cost estimation is not as difficult as theoretical physics or rocket science so, given the technical aptitude of those within the aerospace and high technology industries, one would think that the community would be able to do a better job of cost estimation. This paper explores the contention that (1) many BPs exist but are simply not applied in a consistent manner and (2) the accepted set of cost estimation BPs that does exist, does not completely address the entire cost estimation life-cycle. This paper offers a more comprehensive cost estimation life-cycle and an approach for assuring that BPs and associated activities are followed.

2. State of Current Practice

Software cost growth may occur for several reasons, including requirements changes, new technologies, optimistic heritage assumptions, simultaneous hardware development provided by multiple partners, basic poor planning practices [2, 6], miss application of methods, and corporate 'strategic-pricing' with hopes of profiting on follow-on change requests. Researchers have mostly focused on issues associated with errors in estimation and often seek to identify better estimation methods. Such efforts have resulted in varying levels of support for approaches such as analogy methods [8], nearest neighbor methods

---

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

[9] and regression models [1, 10, 12]. Although, more recent research has proposed support for hybrid methods [11]. Cost Practioners have focused on issues associated with the use of regression models [10], validation of model performance [13] and estimation handbooks. Such estimation handbooks typically describe the basic characteristics of effective estimating which include clear identification of task, broad participation in preparing estimates, availability of valid data, standardized structure for the estimate, provision for program uncertainties, recognition of inflation, recognition of excluded costs, independent review of estimates, and revision of estimates for significant program changes [7, 14, 15, 16]. All of the listed methods and activities should be considered Software Estimation BP[2] .

3. Cost Estimation Best Practices

The following is specific list of software cost estimation Best Practices derived from the reference books, handbooks and research papers discussed above[1, 7, 9, 11, 12, 13, 14, 15, 16] combined with material from the NASA JPL Software Cost Estimation Class [17] the International Society of Parametric Analysts Certification Class [4].

Establish a cost estimation infrastructure

These practices make estimation repeatable, consistent, and provide the basis for objective data driven estimates.

1. Develop and maintain a documented estimation process
2. Establish a tailorable standard cost structure for all projects
3. Develop and maintain a cost database that captures the organization's history. The database should include
    a. effort and cost actuals
    b. history of estimates and budgets over the lifecycle of a project.
    c. key planning parameters such as software size, requirements counts, cost model inputs
    d. planned schedule dates for major milestones
    e. defects counts
4. Develop and maintain cost models that are regularly tuned and validated against actuals
5. Peer review on a periodic basis

Performing the Estimate

These practices enable objective estimates that address the inherent uncertainty in a cost estimate, especially when made during the early stages in the life-cycle.

1. Clearly define and documented the scope of effort
2. Determine the size of software
    a. Most common sizing metrics are lines of code, function points, software modules and requirements
    b. Be conservative with reuse assumptions
3. Base estimates on data whenever possible
4. Develop an integrated estimate which incorporates technical scope, high level decomposition and a schedule
5. Use multiple estimates which typically consists of
    a. bottom up estimate
    b. model based estimate
    c. system/mission level analogies
6. Incorporate model assumption and uncertainty by estimating a cost distribution
7. Identify and include risks in the estimate
8. Review and peer-review the estimate
9. Re-estimate when changes occur and at major milestones

Document the Estimate

These practices make an estimate defensible, traceable and updatable.

1. Maintain information in an electronic form that can be easily modified
2. Document the basis of estimate (BOE), which should include
    a. Statement of Work and Scope
    b. Work Breakdown Structure (WBS) with associated dictionary
    c. Effort estimates with supporting assumptions
    d. Planning parameters or supporting lower level estimates, e.g. software size estimates
    e. Supporting model estimates and analogies
    f. Schedule
    g. Procurements
    h. Acquisition approach (if applicable)
    i. Cost estimates
    j. Significant cost and risk drivers
    k. Risk items, issues, and/or any known liens

[2] As an aside if one were to focus on estimation in the construction industry or for hardware, there would be common elements but also differences as estimation for different types of projects and industries presents different opportunities and risks.

Monitoring Performance

These practices provide baseline and trend indicators of performance. They also identify areas of weakness, the need for process improvement and impact measures.

1. Track performance
   a. Define a set of metrics for monitoring estimate accuracy and budget growth
2. Monitor key assumptions and planning parameters such as sizing metrics.

What is not addressed directly here is the issue that all too often budgets are set while minimizing or even ignoring the estimates of the software teams or organizations. This is partially addressed by 15a, which enables monitoring estimates and budgets against actuals and would provide objective data when unrealistic budgets are a systemic problem in an organization.

4. What is Software Assurance

Given the paradox of maintaining a reasonable industry standards and BPs for performing software cost estimation and yet repeatedly producing unreasonable cost overruns and estimation inaccuracies, one should consider whether or not the software cost estimation BPs are actually being implemented. Efforts associated with assuring that such software cost BPs and standards are followed can be borrowed from the Software Assurance community and its methodologies for assuring software during the software development process. This paper suggests that key philosophies and techniques found within the discipline of software assurance, if applied to software cost estimation efforts, could help to produce higher quality and more reliable software cost estimates. Software Assurance techniques have been proven to increase the quality and credibility of software products [18,21]. Just as software assurance techniques help to better software development efforts and the usefulness of software products, these same software assurance techniques can be extrapolated to increase the quality and credibility of software cost estimation efforts. This section describes some of the fundamental principles of software assurance and offers suggestions as to the specific techniques that could be applied to software cost estimation activities.

Software Assurance is the engineering discipline that ensures quality is built into software processes and products and that the software products operate safely. The NASA, agency-level NASA Software Assurance Standard defines Software Assurance as the planned and systematic set of activities that ensure that software life cycle processes and products conform to requirements, standards, and procedures. For NASA this includes the disciplines of Software Quality (functions of Software Quality Engineering, Software Quality Assurance, Software Quality Control), Software Safety, Software Reliability, Software Verification and Validation, and IV&V [20]. Each sub-function within the over-arching discipline of Software Assurance contributes to identifying and/or mitigating risks and builds credibility into the software development process, such that quality software processes and software products are developed. Although it is recognized that how Software Assurance methodologies are performed may differ from one corporate culture to the next, many government agencies and corporations have developed at least some form of Software Assurance or quality control practices for software development. This paper will mostly focus on the Software Assurance methodologies used at NASA and how these methodologies could be applied to improve the quality of software cost estimates. It is recognized that at NASA, the discipline of Software Assurance includes the sub-disciplines of Software Safety, Software Reliability, Software Quality and Software Verification. However, this paper will mostly offer parallels between Software Quality Assurance and how it can be applied to software cost estimation.

Software Quality

Software Quality Assurance implements techniques to make sure that that software development products and processes are in conformance with established procedures, appropriate for the project of interest. For example, Software Quality Assurance engineers emphasizes that specific quality plans are in place for each software development effort. Such plans may include configuration management plans, risk management plans, software management plans, software development plans, etc. These plans are to be adhered to throughout the software development lifecycle. Typically, the Software Quality Assurance engineer will establish working lines of communication between the software systems engineer and project management leadership to ensure that any higher-level standards and BPs are flowed down to the project-level and that appropriate project-level plans are followed. Another key attribute of the software quality assurance effort is making sure that software requirements are clearly written,, traceable and verifiable. To this extent quality assurance engineers review all plans, procedures, requirements, design, verification documentation, reports, schedules, and records and assess them for risk to the project and impact against the quality of the software being developed. Formal testing events and peer reviews are typically attended by software quality assurance personnel who serve as independent assessors of activities. Software quality audits may be planned (or initiated randomly) to assess the project's adherence to product quality standards and procedures. Software quality assurance engineers assure that software quality metrics have been defined and are being used to ensure that data trends associated with risk areas are reported and mitigated as necessary. Software Quality Assurance engineers assure that software is tested appropriately and verified for compliance with functional and performance requirements. Problem reporting is of high interest to software quality assurance engineers. Likewise, portions of the software that are deemed significant and may add additional risk to the development effort are reported to key decision-making boards and panels. To this extent the role of the Software Quality Assurance engineer is to assure that software is being developed according to the given standards and procedures predefined by the agency and project of interest. These processes and procedures help to establish control, consistency, repeatability, maintenance, sustainability, traceability, and trend analysis. These attributes help to establish robust software development efforts and builds confidence that the software being developed will perform successfully and safely.

Example Design Inspection Walkthrough Checklist

There are several checklists that can be used to perform quality assurance functions during the software development lifecycle. Figure 1 provides an example of a Checklist, used to assure that software design is developed within the context of a quality standard. This checklist was developed by the Software Engineering Division at Goddard Space Flight Center, such that during a design walkthrough, the software might be evaluated for Completeness, Suitability, Correctness, Simplicity, and Quality. It is important to note here, that this checklist was developed by the software engineering division, and not by the Quality Assurance organization, which is a positive sign of internal accountability by the Software Engineering organization and the Software Assurance organization. Additional independent Software Quality checklists may be offered as examples in future papers. The software quality assurance organization at GSFC would use checklists such as this to perform compliance assessments against the software design.

1	**Completeness** – Specification of design is to the l appropriate level	
	Guidance: Not all may be applicable for a particular system (e.g., not all systems will need to consider COTs), but each check should be considered.	
	Review Requirements Traceability Matrix to ensure coverage of all requirements	
	Ensure coverage of:	
	Real-time requirements	
	Performance issues (memory and timing)	
	Spare capacity (CPU and memory)	
	Maintainability	
	Understandability	
	Database requirements	
	Loading and initialization	
	Error handling and recovery	
	User interface issues	
	Software upgrades	
	Software re-use and modifications	
	COTS	
	All inputs and outputs	
	Clearly and correctly identify interfaces	
	All functions clearly and accurately described in sufficient detail	
	All interfaces clearly and (appropriately) precisely defined	
	Adequate data structures defined	
	All error codes documented	
2	**Suitability**– The design itself is good.	
	Deviations from the requirements are documented and approved	
	Assumptions are documented	
	Major design decisions are documented	
	The design is expressed in precise unambiguous terms	
	Dependencies on other functions, operating system, hardware etc. are documented	
	The design follows notational conventions	
3	**Correctness** – The design will lead to good softwa	
	The logic is correct	
	Memory and timing budgets are reasonable and achievable	
	Error messages are helpful and understandable	
	The design is understandable (i.e., easy to read, to follow logic)	
	It is maintainable (i.e., no obscure logic);	
	It is testable	
	It is consistent (i.e., program flow and data format match between sending and receiving components/software units)	
	It is cohesive (i.e., proper groupings of related components/functions)	
	It is mutually suspicious (i.e., the components/software units check each other for errors in parameters or other exchanged data)	
	COTS and GOTS have been verified to fulfill their intended purpose	
4	**Simplicity** – Complexity no more than necessary	
5	**Quality** – A high quality design of a high quality sys	
	Have alternate design approaches been evaluated and the optimum chosen?	
	User interface/screens have been verified with end users?	
	Are there minimal requirements TBD's?	

Figure 1: Design Inspection Walkthrough Checklist

5. How Can Assurance Methods be Applied to Software Cost Estimates

NASA maintains agency level standards/requirements for both Software Engineering as well as Software Assurance. NASA programs and projects must abide by these standards when developing software, or seek relief in the form of a waiver or deviation. These standards impose requirements on procedures, architecture efforts, implementation activities, and other related tasks used to acquire, develop, assure, and maintain software for NASA programs. The Software Engineering agency-level requirements are designed to be a minimum set of requirements to protect the Agency's investment in software engineering products and to fulfill its responsibility to the citizens of the United States of America [21]. Likewise NASA agency-level Software Assurance requirements were developed in order to establish a common framework, including generic quality procedures for the software assurance process in support of all life cycle processes, establish and support the cooperation of various groups who are conducting different aspects of the total software assurance process, support and utilize the independent reporting structure required for NASA safety, reliability, and quality processes, and define software assurance activities and tasks to meet the objectives of software assurance [18]. These two high-level agency standards, provide the overall context for which software engineering and software assurance should be performed. Together, the standards compliment each other in order to comprehensively address the development processes/practices as well as to assure that those processes/practices are fulfilled according to the established plans. This approach can be adopted within the community of software estimators. The following text offer parallels between Software Assurance methodologies and how they could be applied to Software Cost Estimation.

One of the paramount functions within Software Assurance is ensuring that guidelines, standards, processes and procedures are documented and followed. Without the appropriate level of oversight, software development efforts may loose configuration control, traceability, sustainability and maintainability. Software Assurance engineers could extend these very same methodologies to ensure that software cost estimation processes and standards are followed. Software Assurance personnel could maintain configuration control and track any changes to the estimation process. Software Assurance personnel could manage any tailoring of estimation standards or deviations from the defined process as well as facilitate any impact analysis to be performed on any major changes to the process. This approach would enable the appropriate amount of independent over-sight and in-sight, by extending typical software assurance functions to include assurance of software cost estimation efforts.

Software assurance requirements typically require traceability between the software engineering requirements and areas of concern related to safety, quality, and reliability. For example, NASA software developments require that safety critical software requirements are clearly defined as safety critical and are traceable to hazard analysis. To this extent, if there are changes to any requirements that have safety implications, the appropriate parties are made aware of such changes. This requirement traceability is required to ensure that there is an awareness of how the such software changes may impact the over all safety or reliability of the system. Likewise, one might want to trace specific software cost estimates to particular areas of the software development activity, such that any metrics anomalies can be clearly identified and isolated for the appropriate corrective action.

Software Assurance standards typically require the proper training and skill levels for those developing software, testing software and implementing software. This is especially important in the case of safety critical or mission critical software. This training emphasizes the idea is that those developing software should have attained a given level of aptitude or one might expect the resulting software quality to be compromised. Likewise, operating software within a testing environment without the appropriate skill level to do so, could be detrimental to the testing exercise, could impose damage on the system and even more severe, could cause loss of human life. As a result, software assurance engineers levy requirements that ensure software developers, testers and operators are properly trained.

The essence of such training requirements could be applied to personnel performing software cost estimations. The person performing the software cost estimate should be well trained within the industry and discipline of software development for which the estimate is being performed. In other words, the software cost estimator should be familiar with the development for which he/she is estimating. Often times, the cost estimators work within completely different business units or organizations from which the software is being performed. The cost estimator should at least have some domain knowledge of that which is to be estimated.. Maintaining the appropriate level of training also applies to understanding what requirements and standards the software developers will need to follow. For example, NASA standards invoke a variety of requirements depending on if the mission of interest is a human-rated mission, robotic-space mission of high importance or a low-risk experiment. Maintaining a working knowledge of the software standards and requirements can be ensured via the proper training of the software cost estimators. Lastly, training applies to the proper use of any tools used to perform the software cost estimates. Assurance engineers should ensure that software cost estimators are well and trained on the tool being used and that they are knowledgeable enough on that specific tool to use it properly.

Software Assurance engineers are typically involved with ensuring that algorithms and quantitative models used within the flight and ground software meets the intended need of use. Software Assurance engineers ensure that such algorithms and models are implemented correctly and are controlled properly. There are many problems that can result from loose control and open manipulation of algorithms in spacecraft software. Software assurance engineers support the development and planning of such algorithms to ensure that processes are adhered to and testing of algorithms is sufficient. These same software assurance principles should be applied to software cost estimation efforts involving quantitative modeling and manipulation of estimation inputs. Software estimation tools use a variety of different algorithms and modeling assumptions, some of which can be manipulated by the user. Manipulation of estimation modeling algorithms and or the use of equations for estimating cost should be verified and controlled to the appropriate level of intended use by Software Assurance engineers. Software Assurance engineers could independently verify cost models of interest, ensure cost models are in synch with software cost estimation standards and regulatory documents, and ensure models are used within the context of their purpose. When cost estimate distributions are performed, Software Assurance engineers should maintain objective evidence of distribution results for future references and any potential distribution function re-runs.

Software Assurance engineers typically support software development peer reviews activities in order to gain necessary levels of insight needed to perform software assurance functions as well as to ensure that such peer reviews are being conducted according to the standards. This function can be extended to have Software Assurance engineers participate in cost estimation reviews.

The use of historical databases can be an enormous help to software cost estimators. However, when using historical databases it is important to make sure that the information in those databases are verified for the intended use and any known areas of concern are addressed prior to use. Databases may include information from previous projects, however assumptions used for one project may be very different from another project. Software Assurance personnel could help to review software cost databases for consistency with software standards, maintain configuration control and track any major changes with database and audit estimation efforts for proper use of the database.

Software Assurance Engineers and Software Engineers use metric data to expose trends that may lead to problematic portions of the software or risk areas of interest. Metrics should be kept when performing software estimates and compared to actuals throughout the life of the development activity for similar reasons. Such trend data can help to uncover problem or risk areas associated with cost overruns or inaccuracies of a particular group or functional area.

Software cost estimators will typically develop some form of assumption regarding the size of the software. It may help to have Software Assurance personnel provide an independent verification of the sizing estimate, check inputs and assumptions for sizing activity, and identify any areas of concern or any considerations that may not be taken into account by sizing tools. This would offer sort of an independent sanity-check on the sizing estimate and provide the estimators with multiple data points on which to base a final estimate. Software Assurance personnel could help vet the software cost estimate approach with personnel at the working level, who may be responsible for performing the actual software development effort. This would help to ensure that the expectations associated with the software cost estimates are within reason with the personnel expected to perform the actual software development activities.

Software Assurance engineers are involved with every major milestone review throughout the software development lifecycle. The intent is that certain activities need to be reevaluated at least at every major milestone. Likewise, Software Assurance engineers could review software cost estimates at major milestone reviews in order to compare actuals to the initial estimates. This comparison would help ensure that the quality of the estimation practices are not degraded as the project moves forward and it would help to ensure that any new assumptions have been taken into account.

Example Sizing and Heritage Checklists

Software Quality checklists are used as tools for identifying key items of interest during assurance activities. Similar checklists can be used improve software cost estimate quality. Figure 1, provides a notional software cost estimation checklist. Two of the most important cost drivers are the estimates of the size of the system and the amount of reuse (heritage) from previous systems. Given the level of importance that these two cost drivers impose on estimation efforts, a checklist approach might be especially beneficial. These two areas of interest are often root cause contributors for software estimation inaccuracies and cost overruns. Notional examples of a Software Lines of Code (SLOC) Determination Quality Assurance Checklist and a Consideration for Heritage Quality Assurance Checklist that could be used for Software Cost Estimation purposes are provided in Figure 2 and Figure 3.

Software Size Checklist
1. Source of analogy reference (be specific):
2. Why is this analogy appropriate?
3. Is the code count consistent with institutional or project counting rules?
a. Was a code counter used?
b. Delivered vs written code?
c. Types of source lines included (Executable
i. Nonexecutable
ii. Declarations
iii. Compiler directives
iv. Comments
v. Blank lines
4. Are size adjustment rules documented
5. Are size adjustment rules reproducible

Figure 2: Example Software Size Checklist

Software Heritage Checklist
1. Project/System
2. Software Element Name (Modules, CSCI, Subsystem)
3. Software Element Description
4. History of Heritage Software
a) Software Class of Heritage Module (Class A, B, C)
b) Has the software element been successfully reused ona previous project?
c) Was the Heritage Module designed to be reused (Yes, No)
d) Do the following artifacts exist
i) Design
ii) Unit Test
iii) Open Defect Reports
iv) Record of previous failure reports exists
5. Heritage Type (Ruese, Reengineered)
6. New Module Software Class (Class A, B, C)
7. Similarity of Use Case with the heritage element
8. Is the heritage software compatible with current requirements

Figure 3: Example Software Heritage Checklist

6. Lessons Learned from Software Process Assessment Methods at JPL

While the exact method proposed in this paper has not been performed there are two JPL activities being performed that provide some insight into current practice and the usefulness of assessment type methodologies. The process assessment activities that are performed, are the Tailoring Record (TR) used by the engineering divisions and what is known as PPQA or Product and Process Quality Assessments which are typically performed by the software quality assurance organization. The TR is a record of what projects plan to do and PPQA provides a record of what projects are actually doing relative to their intent. The TR captures what the team says they will do based on their self report while PPQA is based on an independent assessment of an artifact [23].

The TR compares the processes used on an individual software project to the institutional Software Development Standard Processes (SDSP). The TR is required because the JPL processes are highly tailorable so that they can be efficiently used by a wide range of software, which differ in size, domain and required reliability. The TR is generated early in the lifecycle of a project or task and ultimately assists the initial writing of software management and development plans. The SDSPs consist of 523 sub-activities in 21 processes3 [24] We only review those processes that are being utilized by the project. For example new flight software developments engage all process while projects in maintenance may only engage the implementation and validation processes. Over the last three to four years approximately fifty TR's have been completed. The initial TRs took 6 to 16 hours to complete. Today as we have learned to make the review process more efficient it only requires 4-6 hours to complete a TR. This indicates that when first starting the TR process it took 1 to 2 minutes per sub-activity while today it requires 0.5 to 0.8 minute per sub-activity. The JPL estimation process has 14 sub-activities which correspond to the Best Practices 6-14 and only at a very high level to Best Practice 16 (See Section 3). As currently written the estimation process takes 7 to 11 minutes to review. At JPL the primary non-performance in the estimation process is documenting the estimate, which is Best Practice 16. To more precisely reflect the Best Practices identified in this paper the number of sub-activities in the current JPL process would need to be increased in length from 50 to 100%. Virtually all the changes would be in the

[3] Excludes the Software Acquisition Management process.

document BOE activity. This suggests that it would take a minimum of 10 minutes to a maximum of 22 minutes to review the expanded process.

PPQA provides objective insight into how well a project is adhering to its planned processes and to the standards it has adopted for its work products. Tasks are required to develop an audit plan, which includes evaluation of processes and products based on the organizations' quality objectives and following the JPL PPQA process. While PPQA can be conducted by line management with appropriate training and oversight, a large percentage of these activities are performed by JPL's Software Quality Assurance (SQA) organization. This evaluation method is more analogous to the assessment method proposed here. Fortunately, very recently the software assurance organization has started to experiment with recording time estimates for completion of different assessment activities. We have small number of observations which indicate it requires from 12 to 20 minutes per page with a 15 minute average. The documents reviewed did not include a planning or BOE document but included requirements, architecture and design documents along with some release documents. These documents ranged in length from 15 to 31 pages with an average length of 19 pages. At JPL BOE's (Best Practices 6 through 16) are typically in slide form not paper style documents. Slide pages are overall less dense then a paper pages. One of the more complete BOE's at JPL was 20 pages with 9 pages of back-up. This suggests an assurance style review would take from 2.5 hours to 7 hours. It would be expected that this would be conducted in preparation to a cost review where the 'correctness' of the estimate would assessed. To date no direct PPQA assessments have been performed on software cost estimates. In the last 2.5 years there has been one finding against estimation that arose during an assessment of a software management plan.

7. Conclusion and Next Steps

Over the last three decades there have been extensive advances in the techniques used to estimate and monitor software development cost. In spite of these improvements in quantitative methods there remains a struggle to complete projects within the planned time and schedule. The essence of this paper is that the community understands the practices and procedures that need to be performed in order to do better cost estimation, it just needs to follow through and do it! One mechanism to increase the use of these known BPs is to use the well-established quality assurance methods of independent audits. This can be performed by the software assurance organization but can also be performed by the engineering or business divisions within an organization. Clearly, this paper does not address all of the causes of software cost growth, yet perhaps adding an assurance function to the software cost estimation effort will reduce the influence of those causes associated with undisciplined and even unprofessional practices.

Future follow-on efforts of this paper include development of a more complete set of assurance checklists, followed by a pilot program to implement these checklists on a NASA project. As part of the pilot study, methodologies would be established to assess and measure the use of the assurance approach and the associated impact on the project's conformance with software cost estimation best practices.

7. References

[1] B. Boehm, Software Engineering Economics. Prentice Hall, 1981.

[2] J. Hihn and H. Habib-agahi, "Identification and measurement of the sources of flight software cost growth," in Proceedings of the 22nd Annual Conference of the International Society of Parametric Analysts (ISPA), Noordwijk, Netherlands, 8-10 May 2000.

[3] Spareref.com, "Nasa to shut down checkout & launch control system," August 26, 2002, http://www.spaceref.com/news/viewnews.html? id=475.

[4] "Certified Parametric Practioner Tutorial,", Proceedings of the 2012 ISPA/SCEA Joint International Conference & Training Workshop, 14-16 May 2012, Brussels, Belgium. Can be found at http://www.sceaonline.org/.

[5] Hihn, J.M. and H. Habib-agahi. Cost Estimation of Software Intensive Projects: A Survey of Current Practices. Proceedings of the Thirteenth IEEE International Conference on Software Engineering, May 13-16, 1991.

[6] Controlling Cost Growth of NASA Earth and Space Science Missions, Committee on Cost Growth in NASA Earth and Space Science Missions, National Research Council, The National Academies Press, 2010.

[7] GAO Cost Estimating and Assessment Guide: Best Practices for Developing and Managing Capital Program Costs, United States Government Accountability Office, GAO-09-3SP, March 2009.

[8] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," IEEE Transactions on Software Engineering, vol. 23, no. 12, November 1997,

[9] M. Shepperd and G. F. Kadoda, "Comparing software prediction techniques using simulation," IEEE Trans. Software Eng, vol. 27,no. 11, pp. 1014–1022, 2001.

[10] C. Kemerer, "An empirical validation of software cost estimation models," Communications of the ACM, vol. 30, no. 5, pp. 416–429, May 1987.

[11] T. Menzies, Z. Chen, J. Hihn, and K. Lum, "Selecting best practices for effort estimation," IEEE Transactions on Software Engineering, November 2006.

[12] R. Strutzke, Estimating Software-Intensive Systems: Products, Projects and Processes. Addison Wesley, 2005.

[13] K. Lum, J. Powell, and J. Hihn, "Validation of spacecraft software cost estimation models for flight and ground systems," in ISPA Conference Proceedings, Software Modeling Track, May 2002.

[14] 2008 NASA Cost Estimating Handbook, NASA Headquarters Cost Analysis Division. Can be found at http://www.ceh.nasa.gov .

[15] Lum, K., Bramble, M., Hihn, J., et. al., JPL Handbook for Software Cost Estimation, JPL D-26303, June 2003

[16] Naval Center for Cost Analysis and Air Force Cost Analysis Agency , Software Development Cost Estimating Handbook, 2008, Can be found at https://acc.dau.mil/CommunityBrowser.aspx?id=323892

[17] Hihn, J., JPL Software Cost estimation Class, JPL, 2012. Cleared version is available by contacting the author jhihn@jpl.nasa.gov.

[18] Port, D., Wilf, J., A Study on the Perceived Value of Software Quality Assurance at JPL, Proceedings of the 44th Hawaii International Conference on System Sciences – 2011, Kauai, Hawaii, January, 2011.

[19] Barry Boehm, LiGuo Huang, Apurva Jain, and Ray Madachy, The ROI of Software Dependability: The iDAVE Model, IEEE SOFTWARE, May/June 2004

[20] NASA Software Assurance Standard NASA-STD-8739.8 2004

[21] NASA Software Engineering Requirements, NPR-7150.2A, 2009

[22] NASA Design Inspection/Walkthrough Checklist, Number 580-CK-058-02, 580 Software Engineering Division (SED), 200X.

[23] J. Hihn, S. Morgan, et. al., JPL Mission Software: State of Software Report 2011 External Release, JPL D-29115. Contact jhihn@jpl.nasa.gov for a copy.

[24] J. Hihn, S. Lewicki, S. Morgan, Bootstrapping Process Improvement Metrics: CMMI Level 4 Process Improvement Metrics in a Level 3 World, Proceedings of the 44th Hawaiian International Conference on System Sciences (HICSS 44), Lihue, HI, January 4-7, 2011