FLIGHT SOFTWARE TRENDS AND PATTERNS IN THE AEROSPACE INDUSTRY: JPL LESSONS LEARNED

Jairus M. Hihn
Karen Lum
John Powell
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

Abstract—As in industry, JPL has experienced numerous changes in how we develop both flight and ground software. Software languages, design methods, and development processes have all changed from what they were in the early to mid 80's. It used to be that our spacecraft were flying hardware with a computer on board. Today our spacecraft are more and more becoming complex flying computers especially with the advent of sophisticated fault protection and auto-navigation software. Because software is playing a more critical role in our deep space missions, there has been a significant increase in interest in software at JPL.

As part of this new focus a number of activities have been initiated to improve how we manage and how we measure our software activities. Currently we are validating and calibrating commercial parametric tools, as well as developing our own models. As a result of integrating our cost databases and engaging in an extensive software metrics activity, it has become possible to analyze JPL's historical datasets for trends in metrics. In this paper, we will summarize the software trends and their impact on the cost of developing flight and ground software.

INTRODUCTION

There has been an increase in interest in software at JPL because software is playing a more critical role in our deep space missions. Recently the Software Quality Improvement (SQI) Project has been formed to achieve and sustain excellence in software engineering at JPL to enable mission success. It will enable and promote software best practices, and leverage JPL experience in software engineering in support of major software projects, throughout the entire software life cycle. The goal of the SQI Project is to establish an operational software improvement program that results in the continuous measurable improvement of software quality at JPL. Its objectives include improving cost and schedule predictability, improving the quality of mission-critical software, reducing software defect rates during testing and operations, increasing software development productivity, promoting software reuse, and reducing project start-up time.

As a result of the recent integration of our cost databases, in support of the SQI project, it has become possible to analyze the historical datasets for trends in software development cost, productivity rates, as well as some schedule and quality-related metrics. These analyses are intended to support software managers by providing a quantitative basis for informed decision-making regarding their projects. Further, the value to JPL of the trend analysis is expected to improve as the lab engages in an extensive data collection activity over the next few years. In conjunction with historical data trend analysis, there has been a successful ongoing effort to validate and calibrate commercial parametric tools such as COCOMO, SEER-SEM and Price S as well as developing our own models in an effort to improve cost estimations for software projects at JPL. There is also a major focus on developing a JPL version of the COQUALMO model to provide predictive defect introduction and removal estimates to support ongoing quality assessment and planning throughout the software lifecycle.

In this paper, we will summarize our activities as well as various software trends at JPL and their impact on the cost of developing flight and ground software. The various trends to be discussed include:

- Effort to size relationships
- Various productivity trends
- Size and effort trends over time (past 3 decades)
- Defect rates (actual and predicted) as they
pertain to schedule and effort. These relationships will also be compared and contrasted across to major software categories at JPL, specifically flight software and ground software.

**DATABASES AND METHODOLOGY**

The data used in this study is the integration of data from various sources that has been collected by JPL's Cost, Risk & Systems Analysis Group since 1986. The major sources for the data are:

- **NASA Software Cost Database (1986 - 1990)**
  Consists of 100 ground and 20 flight data points at subsystem-level from all NASA Flight Centers, collected post delivery

  Contains 49 data points at assembly-level, collected at time of delivery

  Contains over 15 DSN subsystem upgrades, collected at Preliminary Design Review and each major delivery

- **JPL SQI Foundation Measures Database (2001 - on-going)**
  Consists of 8 JPL flight software subsystems
  Defect data from 4 JPL projects

The majority of the data was collected to support a JPL version of the COCOMO software cost estimation model. The NASA, SORCE and DSN software cost databases were collected using a survey instrument to collect inputs for the COCOMO 81 version of the model.

This was supplemented to obtain additional information on schedule and effort allocation across phases and activities. Obtaining this information typically required interviews with software managers and technical personnel ranging from 1 person for 1-2 hours per record in the NASA database to interviewing from 3-6 persons for 1-2 hours with an additional 1-2 work weeks in analyzing the software projects' schedules, estimates, and official budgets. This data was collected from 1986 through 1997.

In 2001 when the SQI Measurement Program was initiated, data was collected to support the use of COCOMO II and SEER-SEM for cost estimation and a range of software engineering models for establishing baselines for planning and defect analysis. There exist a significant overlap between the input data required for COCOMO II and SEER-SEM. Therefore, questionnaires that were originally designed for COCOMO II input data collection were reviewed and adapted for use in an effort to collect SEER-SEM data simultaneously. The revised survey was subsequently used in interviews with key software people of the various projects. The similarity of SEER-SEM and COCOMO II parameters/cost drivers facilitated the ability to map survey answers (data) to both models.

The survey instrument asked the key software people to rate their piece of the project in various categories, from team capabilities, software reliability, complexity, and tool usage. Multiple (2-3) interviewers participated in each of the interviews conducted. Then interviewers compared notes taken during the interviews and cost driver ratings derived from conversation with the interviewee. All cost driver-rating discrepancies were resolved through discussion between the interviewers. Discrepancies were noted and, when appropriate, ranges were formed for input parameters when full resolution could not be reached. Using the results from these interviews along with historical data, the models' inputs were entered into the associated implementation tools (software) for each. Follow-up interviews were conducted as necessary for the purpose of data conflict resolution and clarification with regard to project scope and CSCI applicability. Past projects were selected in order that actual cost and effort data could be collected to compare with the estimates produced by the models.

Some data had to be scrubbed. For example, different projects include different activities and life cycle phases in their effort actuals. To make the records comparable, adjustments were required. To discount reused and inherited code, equivalent lines were estimated to get an indication of the actual number of lines written and modified.

This yielded a total of 190 data records covering ground and flight software implementations that completed from 1980 through projects that will not be completed until 2008. The datum included in the integrated database are start date, completion date, equivalent source lines of code (physical lines, no comments, no blanks), effort in work months, schedule length, application category (flight or ground), and whether the data reflected actuals or estimate-to-complete. Not all records have data for all fields.

Due to gaps in data collection over the years there are gaps in the records, such that there are no ground software records post 1997 and no flight software records from 1990-1995.
COST PREDICTION MODELING

As part of the SQI Project's goal to improve cost predictability, a study to evaluate three models – COCOMO II, SEER-SEM, and PRICE S – was initiated in 2001. JPL, because its primary focus is developing and operating deep space science missions, has many characteristics that make it unique from other organizations that develop software. At the same time, there can be many things in common. Therefore, the study was started to determine whether some cost estimation tools could be used “out of the box” or even if they were applicable to the JPL environment at all.

These models have been validated for both flight and ground software development at JPL using historical data consisting of 10 flight software projects and 9 DSN ground software projects. The Post Architecture COCOMO II model, SEER-SEM, and PRICE S have been assessed “out of the box” and predict software costs reasonably well in the JPL environment.

All models predicted better for flight software than ground software in general. COCOMO II had strong results for both flight and ground software. SEER predicted well for flight software but not as well for ground. (Figures 1 and 2) A validation of the SEER-SEM model using only its knowledge bases for input parameters was also performed. It was found that using only the knowledge bases for performing estimates did not correspond well to the JPL environment. PRICE S was the strongest predicting model for flight software and ground software. However, its prediction range is wider than COCOMO II's.

It was found that all three of the “uncalibrated” models being evaluated – COCOMO II, SEER-SEM, and PRICE S – were able to predict within similar ranges based on the measures we used to evaluate the models. On average 50% of the model estimates predicted within 30% of the actuals. Given that these models were unadjusted for JPL's local environment, they performed much better than originally expected.

Although the models predict within a reasonable range, it is our goal after adjusting the models – by either calibration or some other consistent method – to get 80% of the estimates within 30% of actual effort. Calibration of the models will require collecting additional data. The performance is expected to improve as more data is collected. This data collection activity will not only enable more precise calibration of the software cost models, but will enable the analysis of historical trends.
Comparison of Uncalibrated Models' Accuracy for Ground Software

Figure 2. Uncalibrated Ground Software Accuracy

COST METRICS TRENDS

Software Size and Languages

There is an expected trend of significant software size growth over time for flight and instrument software as observed in Figure 3. While ground software shows a decline in average size, it is not statistically significant. The corresponding time period also shows a migration towards C and C++ from other programming languages in general. (Figure 4) Fortran is decreasing in use for development somewhat but still remains in significant use. This appears to be due to the persistent amount of software being maintained that was written in Fortran. (Figure 5) While C and C++ are the language of choice for software development at JPL, critical software functionality, implemented in Fortran is essential to many missions and software needs.

Productivity Trends

Productivity trends have been summarized using two different perspectives. The first (Figure 6) shows just the average productivity (SLOC/work-month) for flight, instrument, and ground software for the eighties, nineties and 2000-2008. Here it can be seen that productivity has been increasing for all software and quite dramatically for flight software over the past twenty years. Between the eighties and the nineties, ground software productivity only increased by approximately 36%, while flight and instrument software increased productivity by 55%-63%. This is most likely a reflection of the huge increase in computing capacity of flight-qualified processors and the move to the use of modern programming languages such as C instead of Assembly. In the eighties, most of the flight software was written in Assembly while in the nineties there was a shift to C and Ada, which became
possible with the more powerful processors. Estimates for the current decade indicate productivity for flight software will decrease. This could be misleading, as software size tends to be underestimated. However, it also could be a reflection of the shift towards higher-quality, more reliable software and away from faster, better, cheaper, which, with hindsight, clearly reduced software reliability and increased risk.

Figure 7 is another way to look at productivity trends. Displayed is productivity against total effort in work-months divided by schedule length. This is an indicator of how many people have to coordinate on average over the life of the project. It can also be used as an indicator of schedule compression, trying to get too much work done in too little time. Five different relationships were derived, Ground 80's, Ground 90's, Flight 80's, Flight 90's, and Flight 2000+. In all of these relationships it can be seen that as development effort over schedule becomes tighter, development becomes less productive. As time increases, productivity versus compressed effort has been increasing, for both flight and ground software, although flight software in general has a lower productivity versus effort over schedule than ground software.
**Size versus Effort**

Both ground and flight software show trends of moving from decreasing to increasing returns to scale from the 1980s to the 1990s. (Figure 8) This shift can stem from the change in characteristics of software development over time. In the 1980s, there was little reuse and no auto code generation. Most programming was done in languages such as Assembly, and Fortran, with lots of declarations. The result was that a line written tended to be a line delivered. In the 1990s object-oriented languages and CASE tools allowed for greater inheritance and reuse.
In addition, most of the flight projects in the 1990s consisted of projects that may not have properly adjusted for reuse. Although software reuse has increased over time, it is difficult to sort out. The exponent on the Size variable (Table 1) may not necessarily be less than 1.0 in terms of true equivalent lines of code.

Table 1. Effort Equations

<table>
<thead>
<tr>
<th>Flight 1980s</th>
<th>Equation</th>
<th>R-squared</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flight 1990s</td>
<td>$E = 8.2S^{1.15}$</td>
<td>0.68</td>
</tr>
<tr>
<td>Flight 2000-08</td>
<td>$E = 7.8S^{0.98}$</td>
<td>0.55</td>
</tr>
<tr>
<td>Ground 1980s</td>
<td>$E = 4.7S^{1.01}$</td>
<td>0.91</td>
</tr>
<tr>
<td>Ground 1990s</td>
<td>$E = 8.9S^{0.79}$</td>
<td>0.56</td>
</tr>
</tbody>
</table>

$E$ is effort; $S$ is Size

As expected, the fixed costs for flight software was higher than ground software in both the 1980s and 1990s. The fixed cost of software development, represented by the constant term, has increased from the 1980s to the 1990s for ground software. This increase in fixed cost may suggest that the reduction in marginal cost is justified as the same pattern can be seen for flight software. Given that there were so few data for flight software in the 2000-08 time period, there is no significant evidence for a fixed cost decrease in the 2000s.

The bottom line is that no matter how one looks at the software metrics, there has been a marked increase in development productivity.

**DEFECT PREDICTION MODELING**

One of the approaches that is currently being explored at JPL for defect prediction with regard to defect densities involves the adaptation and calibration of the COQUALMO model for use in the JPL environment.\(^2\) The COQUALMO model is an extension to the COCOMO II model that seeks to predict software defect densities based on early lifecycle characteristics.\(^3\,4\,5\,6\) The model is regarded as an experimental portion or the Software Quality Improvement (SQI) Project's measurement and benchmarking activities within the larger framework of the Metrics program at JPL. These characteristics include a subset of the input data needed to perform cost estimations using the COCOMO II model, namely 21 of the 22 cost drivers. In addition to the COCOMO II cost drivers, three defect removal profile ratings are collected from software projects. The defect removal profiles describe the defect removal activities to be performed during a given software project and the degree of rigor with which they will be applied. The COQUALMO model then produces a prediction of the introduction and removal defects as well as the residual delivered defect density. The current COQUALMO model is based on expert opinion though two rounds of Delphi analysis conducted by the Center for Software Engineering (CSE) at the University of Southern California (USC).\(^7\,8\)

**Defect Prediction Model and Methodology**

COQUALMO consists of two independent sub-models 1) the Defect Introduction Model (DIM) and 2) the Defect Removal Model (DRM). Twenty-one of the 22 COCOMO II cost drivers are used by the DIM along with internal baseline defect discovery rates to produce an estimate of the number of defects that will be introduced during the development of the software in question. The COCOMO II drivers and defect removal profiles are collected from past and ongoing projects via interviews with project personnel. The estimate categorizes the defects into three categories of introduction:

- Requirements
- Design
- Code & Test

The results of the DIM are used as input to the DRM along with three defect removal profile ratings. Each defect removal profile represents a different classification of defect removal activity:

- Automated Analysis Activities
- Peer (People Oriented) Review Activities
- (Traditional) Testing Activities

The rating for each of these profiles refers to the degree of rigor used in each classification. For example in the Peer Reviews categories, the rigor can range from no peer reviews at all (Lowest Rating) to full Fagan Inspections (Highest Rating). Each profile rating will impact the number of defects removed by estimating that a given percentage of the defects from the DIM estimate are removed. Each percentage associated with each selected rating is applied to the remaining defect in turn. (See Example 1)

COQUALMO also distributes the removal of defects over the defect categories accordingly and displays defects introduced/removed/remaining for each category of software being estimated.

The estimates produced by the COQUALMO model are compared to actual defect introduction and removal
rates for the same categories. The actual defect rates presented in this paper were collected from past projects via JPL defect tracking system archives and interviews with project personnel for verification and clarification as needed.

IF

It is estimated that 100 defects will be introduced and defects are removed at the following rate
Automated Testing removes 20%
Peer Reviews remove 30%
Testing removes 60%

THEN

The residual number of defects delivered is Calculated as

\[ 100 \times (1 - 0.20) \times (1 - 0.30) \times (1 - 0.60) \]
\[ (100 - 20) \times (1 - 0.30) \times (1 - 0.60) \]
\[ 80 \times (1 - 0.30) \times (1 - 0.60) \]
\[ (80 - 24) \times (1 - 0.60) \]
\[ 56 - (1 - 0.60) \]
\[ 56 - 33.6 \]
\[ 22.4 \]

Example 1: Defect Removal Calculation

Finally, the data and predictions are analyzed in an attempt to identify consistent trends and correlations between defect predictions for projects and the project's actual defect introduction and removal patterns. The identification of such patterns aids in the adaptation and calibration of the defect prediction model to the JPL environment, as well as providing valuable trend information regarding defects to ongoing projects in the process.

Preliminary Defect Prediction Results

The preliminary results from the JPL efforts to model defect introduction and removal from a predictive standpoint involves the characterization of projects through COCOMO II cost driver descriptions and defect removal profile ratings. Currently, the base of data is small (4 data points) because the efforts towards the goal of defect density prediction are in their early stages. However, the ongoing effort is continuously providing more data and a statistically significant sample is expected in the near future.

Under the caveat that the database is small and assuming a yet to be proven hypothesis that the current trends will continue, various trends with regard to the defect prediction model can be discussed. First, the model appears to reasonably discriminate defect types (requirements, design and code & test) in a manner consistent with phenomenen observed in actual historical defect data from past projects. (Figure 9) Next, there appear to be early indications of different levels of defect introduction and removal rates between flight software and ground-based software. (Figure 10) Finally, from Figure 9, it can be seen that the defect prediction model over-estimates the percentage of defects that will be introduced in requirements and design phases, while underestimating the number of defects that will be introduced in the coding phases. However, a consistent trend towards increased defect rates, as software development progress from one phase to another in the life cycle is readily observable. The discrepancies may be due to one or more various factors:

- Inaccuracy in the model's prediction
- Projects neglecting to fully document early lifecycle defect in the process of discovering and removing them
- Mischaracterization of defects as sets of coding defects that actually result from trying to cope with flawed designs or requirements.
- Difficulty in isolating defect introduction in early lifecycle stages (requirements and design) relative to later stages such as coding due to differing availabilities of mature tools and techniques and the fluid nature of early lifecycle decisions

![Figure 9. Defect Percentages by Category](image)
Once it is properly calibrated, the defect prediction model can provide predictive metrics to a manager with reasonable estimates of resources needed to ensure the quality of the software, as well as to provide direction regarding the timing of the need for those resources and in what proportions. For example, if the information in Figures 9 and 10 are viewed together, along with average effort needed to repair defects, the overall resources needed to address defects can be effectively planned and employed during phases (requirements design code & test) where they are most likely to yield the most benefit in terms of software quality.

Figure 11 illustrates an early view of the defect introduction rate as a function of schedule compression. Schedule compression, in this case, refers to the degree of effort that is expended over a period of time. That is to say that, if “high” amounts of effort, on average, are expended per month of schedule time, the schedule is said to be more compressed than projects where the average effort expended per schedule month is lower. Many parametric software models offer direction regarding optimal schedule in terms of risk to on-time completion and budgetary constraints and staffing. The analysis suggested by Figure 11 relates schedule compression to the technical (as opposed to programmatic) risks. The preliminary trend in Figure 11, made up of both flight and ground software, suggests that defect rates are positively correlated with increasing degrees of schedule compression. The data in Figure 11 may also be used to formulate metrics from the historical and predictive data for the purpose of resource planning. The defects-to-schedule-compression relationship affords a manager the quantitative information necessary to make informed decisions about resource planning.

The examination of defect metrics, in conjunction with other metrics such as cost, effort, schedule within an integrated metrics program facilitates improvement through recognition of trends in software process and management activities that must be considered together with quality to produce quality software on time and within budget.

**CONCLUSIONS**

As JPL evolves its software measurement program and develops a range of estimation, planning, and quality assessment models, it is expected to have a growing impact on estimation accuracy and software quality. Having access to historical data has allowed us to jump-start the JPL software measurement and modeling program by developing baseline engineering, planning, and quality models and identifying long-term trends. It would have required years to develop these models had we started from scratch.

Integrating the historical databases presented difficulties in that the data was collected differently and the datum were defined differently with respect to how equivalent lines of code were defined or what effort should be included or left out of development effort.

As the new software measurement database is developed and populated, it too will present challenges in that the metrics are collected to meet multiple objectives presented by the need to develop cost
estimation models, cost management models, and defect models to improve process status and improvement objectives.

ACKNOWLEDGEMENT

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.

REFERENCES


Karen Lum is involved in the collection of software metrics and the development of software cost estimating relationships at the Jet Propulsion Laboratory. She has a MBA in Business Economics and a Certificate in Advanced Information Systems from the California State University, Los Angeles. She has a BA in Economics and Psychology from the University of California at Berkeley. She is one of the main authors of the JPL Software Cost Estimation Handbook. Publications include Best Conference Paper for ISPA 2002: “Validation of Spacecraft Software Cost Estimation Models for Flight and Ground Systems.”

John D. Powell holds a M.S. in Computer Science from West Virginia University and is a software quality assurance researcher at the California Institute of Technology’s Jet Propulsion Laboratory (JPL) in the Quality Assurance office. Currently he performs research in the area of Quality/Cost Estimation and Prediction as well as Formal Methods research for efforts at JPL. Prior to his work at JPL, John worked as a System Software IV&V Analyst for NASA’s prime IV&V contractor (Titan-Averstar) performing IV&V analysis on the Redundancy Management and Control systems for the Space Shuttle’s Checkout Launch and Control System (CLCS). Prior to that, at the NASA Goddard IV&V Facility, John performed research under the Intelligent Systems Initiative exploring alternatives to traditional model checking in conjunction with West Virginia University’s Software Research Laboratory (SRL).