How Spreadsheets Get Us to Mars and Beyond

Abstract

Spreadsheets, spreadsheets everywhere and nary a page of documentation. JPL is NASA’s prime center for deep space missions. In all of our missions spreadsheets have played a major role in managing parts list, managing requirements, monitoring progress, planning budgets, developing the initial concept designs, and the backbone of our infrastructure. In this paper we will share our lessons learned in building various spreadsheet intensive systems and applications. Based on our experience in developing and using these various systems we will propose a number of exploratory ideas as to the dimensions of spreadsheet system complexity. In addition we will propose approaches to documentation, review, and verification of these types of systems.

1. Introduction

Spreadsheets are indeed ubiquitous through out most large organizations. At most companies, a spreadsheet application is provided as part of the basic business or IT software platform, which makes the use of spreadsheets basically free to employees. Spreadsheets are extensively used by individuals to do simple accounting tasks, to track simple lists with one or more characteristics, and to do simple analysis and chart generation. As a result of these factors, everyone in a management or technical position is very comfortable with spreadsheets and the inherent mental model they provide for working with data. Furthermore, it is human nature to resist learning some new fangled interface or tool when the IT department or the process geeks attempt to foist a new and ‘better’ way to do business. Hence, it is not surprising as new organizational information problems arise that the boundaries of spreadsheets get pushed to the limits as people build on what they know.

At the Jet Propulsion Laboratory (JPL), spreadsheets are used to varying degrees in virtually every aspect of our engineering, IT, business and process oriented systems. JPL is a Federally Funded Research and Development Center managed by the California Institute of Technology for the National Aeronautics and Space Administration (NASA). JPL currently has 19 spacecraft and seven science instruments conducting active missions. All of these are part of NASA’s Vision for Space Exploration, designed to explore Earth and space and to send robots and humans to explore the moon, Mars and beyond. In all of these missions spreadsheets have played a major role in managing parts list, managing requirements, monitoring progress, planning budgets, and developing the initial concept designs. Spreadsheets also play a major role in all aspects of our infrastructure. As spreadsheets become more and more an integral part of larger systems the question that arises is when should we start treating them like software. When should spreadsheets be required to have formal requirements and rigorous review and testing.

In the abstract, the major factors that drive the need for process rigor should be the same for spreadsheets as they are for any software system. Therefore, the first question that must be addressed is what is the required reliability of the system? Or alternatively, what is the impact of system failure? At this point it does not seem that the determination of required reliability for a spreadsheet intensive system is any different for any software system.

The other major consideration is spreadsheet system complexity. This is clearly an area that has research potential. The more complex the system the harder it is to review and test and the greater the need to fully understand the system.

In the remainder of the paper we will explore the dimensions of spreadsheet system complexity and share our lessons learned in building various spreadsheet intensive systems and applications. In addition we will propose approaches to documentation, review, and verification of these systems. The following spreadsheet development and/or operations case studies will be documented: spreadsheet centric metrics repository, measuring and enabling infusion of best software practices, managing CMMI appraisals, and cost estimation models.

2. Spreadsheet Complexity

While there is an extensive body of literature that explores system complexity [references] spreadsheets provide some unique considerations at least with how they combine various system features. Spreadsheet
complexity can be divided into computational complexity and interface complexity. At this point we have left out procedural complexity as we do not think it is a major driver for spreadsheets in that the programming paradigm imposes a basic procedural approach that does present some serious difficulties at times but it cannot be altered unless when does not use a spreadsheet at all for that part of the application. Other authors may disagree and this just may be an interesting dialogue.

Computational Complexity
- Straightforward cell based computations vs extensive use of macros
- References across multiple Spreadsheets (closely related to model to model interface)
- Monte Carlo techniques
- Linear vs non-linear equations
- Equation systems vs single equation

Interface complexity needs to account for
- Human spreadsheet interface
- Multiple workbooks
- Spreadsheet-database interface,
- Spreadsheet-applications interface
- Spreadsheet-system interface

Procedural Complexity
- Named cells or vectors vs location referencing

>> In the final paper we will clean up and expand these topics <<

3. Spreadsheet Centric Metrics Repository

Over the last seven years JPL has increasingly realized the need to be able to make quantitative based decisions at both the strategic and tactical management levels. In response, we, as good software engineers developed an operations concept to describe such a system and how it would be used. From the beginning, it was clear that there would be numerous human and technical interfaces because there were a number of commercially supported and home-grown systems for managing the programmatic and technical aspects of our projects and there was little consistency in how they were used. When we began, spreadsheets were nowhere in the fuzzy picture that was beginning to take form.

The concept of operations document was reviewed by potential users of the proposed metrics system and their feedback was incorporated in the final document. There were two concerns mentioned by numerous reviewers. The first was that they did not want to enter numbers into a form, i.e., they wanted us to obtain the data automatically from tools that they use. The second comment was summarized by one reviewer as “What’s in it for me?”. We knew we had to address both of these user concerns.

Automation was obviously important but as we probed the input user interface, it became increasingly clear that a spreadsheet interface to the metrics system would meet many of their needs. Therefore, as SMART began to take form, one of its major interfaces was the lowly spreadsheet.

SMART is the acronym for the Software Measures Analysis and Reporting Tool (SMART), a system that helps a software manager plan a task, track and communicate the status of the task, and make decisions. It also supports the improvement of processes and procedures. SMART consists of a metrics repository, a management dashboard, and planning and cost models. Figure 1 shows an overview of the SMART system. Early on, we decided to use institutional standard procedures in the development process. In addition to the concept of operations document, we wrote a requirements document, a software management plan, and various design documents.

![Figure 1: SMART System Overview](image)

For our initial releases, the first user concern, mentioned previously, was addressed by providing a spreadsheet interface to the system along with a pushbutton that would automatically upload the metrics data to the SMART Repository. Since our users are so comfortable and experienced using spreadsheets, the second user concern, this proved to be a satisfactory answer. Full automation of gathering metrics data from institutional tools will be provided in an evolutionary fashion. The second concern was addressed by providing a management dashboard in a spreadsheet that updates automatically when new metrics data is entered in the spreadsheet. These features were used as part of the CMMI Level 3
assessment obtained in 2007. The current SMART product is the result of an evolutionary process as shown in the following table:

<table>
<thead>
<tr>
<th>FY02</th>
<th>FY03-06</th>
<th>FY07-09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process:</td>
<td>Manual</td>
<td>Direct and automated</td>
</tr>
<tr>
<td></td>
<td>collection procedure of</td>
<td>collection (with use of</td>
</tr>
<tr>
<td></td>
<td>metrics data at conclusion</td>
<td>spreadsheet) for</td>
</tr>
<tr>
<td></td>
<td>of a software product</td>
<td>both metrics data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>collected at major</td>
</tr>
<tr>
<td></td>
<td></td>
<td>milestones and data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>collected monthly during</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the development life-cycle</td>
</tr>
<tr>
<td>Repository:</td>
<td>Spreadsheet</td>
<td>Prototype database for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEsA use only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMART: an online database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>with a Manager’s Dashboard in spreadsheet</td>
</tr>
</tbody>
</table>

Figure 2: Overview of Capabilities Development

The current status of features provided by SMART includes:
- Built using a database and workflow tool
  - Workflow, templates, and triggers
  - Different lifecycles and types of measures (Milestone and Monthly)
  - Both web-based and client interface provided
  - Import/export of measures using spreadsheet interface
- Data organized hierarchically by program and project
  - ~250 pieces of software in SMART repository
- Prototype Management Dashboard
  - Creates 12 charts in a spreadsheet that are automatically updated
  - Pushbutton to save data to SMART Repository
  - Charts support all CMMI tasks
- Automated data collection supporting
  - SLiC (code counter tool)
  - PRS (defect system, in process)

4. Managing Software Quality Improvement

Though JPL is known best for the hardware associated with robotic missions to Mars and other planets, the software written to support the operations of those missions is as critical.

This software includes every domain: from the software that supports science instrument functionality to the flight software that controls the spacecraft to the ground software that sends commands to the spacecraft and instruments and also acquires and processes the data sent back to Earth.

With each new mission, the amount of associated software and its underlying complexity has increased. This has caused the risks associated with the success of these missions to increase such that software is as mission critical as the hardware it runs on.

In response, the Software Quality Improvement (SQI) project was established at JPL in 2002 in response to the recognition of improving software engineering practices across the laboratory. An improvement strategy has been defined and executed based on industry best practices championed by the Software Engineering Institute (SEI) of Carnegie Mellon University. The implementation of this strategy also follows a proven Organization Change Management (OCM) model. Spreadsheet applications were frequently utilized in the activities associated with this implementation.

4.1. Infusion of Software Best Practices

The use of spreadsheets to establish, monitor, and control the infusion of these software best practices across the organization has been extensive. SQI has developed a system where each of these spreadsheets can be coupled to provide quantitative views into the quality of the software being developed at JPL in each domain.

The rolling out and use these spreadsheets followed a simple progression:
1. Software Inventory. An established and maintained list of software products and projects at JPL which included data on software classification/criticality, implementation status, lines of code count, primary and secondary languages used, effort in work years, and other characteristics.
2. Work Product Checklist. Associated with each the software products and projects, a checklist which captured the types of documentation that were generated and the tools that were utilized on the project.

>> The remainder of the write up for section 3 will address the interface complexity issues between Excel, other system repositories and SMART, human interface issues, documentation and the extensive verification methods being used. <<
3. Tailoring Record. This compares the processes used on an individual software project to the institutional Software Development Standard Processes (SDSPs) that have been established at JPL. The SDSPs are traced back to JPL Software Development Requirements, Design Principles, and other laboratory policies and standards. The SDSPs are also traced to NASA Processes and Requirements (NPRs) for System Engineering, Software, and Safety.

The last spreadsheet involves interviewing each software task manager to obtain detailed information on each activity performed on the project. During the interview process, The SQI representative would also provide to the task manager information and education on how to use the institutionally provided tools and work aids which accompany the SDSPs.

Spreadsheets were clearly the best implementation to capture all of the data associated with monitoring and controlling the infusion of software best practices. In addition, all of the spreadsheets described above are interconnected and interface to databases and analysis tools. This allowed the ability to easily generate reports that could be supplied back to the task managers for their own use.

The following charts give a few simple examples of the presentation of data provided by rolling up the software inventory spreadsheets:

A disadvantage in the use of spreadsheets came about in performing the tracing from Tailoring Records to SDSPs and from SDSPs to other standards and requirements. The two dimensional nature of spreadsheet do not inherently allow them to be easily used to establish and maintain one to many or many to
one traces. The verification of these bi-directional traces became very labor intensive and error prone. Ultimately the spreadsheets for the Tailoring Records and requirements were imported into a requirements tracking tool that supported bi-directional tracing.

4.2. Optimization of CMMI Appraisals

Concurrent with the pursuit of overall software improvement, SQI has also engaged in an effort to use the SEI Capability Maturity Model Integration (CMMI) to assess the mission critical software at JPL. The CMMI provides a formal methodology to appraise an organization and establish an industry-recognized Maturity level which is then published by the SEI.

Inherent to this appraisal methodology is the use of spreadsheets referred to as Practice Implementation Indicator Description (PIID) forms. A PIID captures references to the project artifacts associated with a particular practice in the CMMI model and records the characterization which measures the degree to which that practice has been implemented.

To accompany the PIID forms, JPL developed in house multiple databases and analysis tools to manage the artifacts needed for an appraisal and to measure progress and effort.

The project artifacts were stored in simple SQL databases and scripts were developed to verify that each artifact referenced in the PIID existed in the database and vice versa. Additional spreadsheets were used to track action items arising from missing or bad references.

Similar to the system used for monitoring and controlling infusion of software best practices, a system of coupled spreadsheets provided simple methods to track the progress toward an upcoming formal CMMI appraisal. But similarly, there were also limitations due to the inability to perform one to many and many to one relationships.

Performing a formal CMMI appraisal involves poring through often hundreds of artifacts for each project. We attempted to establish hyperlinks from the PIID to the document artifacts in our databases. Unfortunately the spreadsheets could not support multiple hyperlinks in an individual cell. Also the hyperlinks became unstable as the environment changed.

5. Cost Estimation Models

Virtually all cost models at JPL are built in Excel. Four examples of the types of models developed and supported by various parts of JPL are described below.

SCAT (Software Cost Analysis Tool) is a Monte Carlo version of COCOMO implemented as a multi-sheet model that can import from a separate sizing tool. Estimation accuracy is validated and documented [4]. Usability and user error have not been addressed. However, to reduce user error the model is pre-populated with ranges from historical JPL missions. This way a user only has to actually modify a small number of the model parameters. There is an externally released version of SCAT that can be obtained from the Authors.

The Flight Software Cost Model can run stand alone or integrated into Team X tool set. This is a fairly complex multi-sheet model that takes high level system descriptors (pointing accuracy, number of instruments, etc.) estimating SLOC and effort multipliers and then executes a third COCOMO based model (point estimate). In the Team X environment it is part of an integrated multiple workbooks that pass parameters over the network. This model was rigorously verified with extensive documentation as part of JPL’s Concurrent Engineering Team (Team X). In blind test, two different estimators had to produce estimates within 10% of each other using the same high level mission specification. The Team X models (20 subsystems) are the only ones were user error is evaluated in any form.

Each section that builds a major component of a space mission has its own spreadsheet based Space Mission subsystem level Grass roots models. These models tend to be very detailed with a very large number of inputs. If they or a simpler version of these models are used in the Team X environment then they are verified in a similar manner to the Flight Software Cost Model. Only the most aggregated level of an estimate can be verified because the detailed historical data does not exist to verify at the module or element level. This has lead to a running debate between the cost engineers and the rest of the engineering community as they prefer very detailed models so they can map into their design parameters and so they can also refine their estimates. The need to reason about the cost of different programmatic and mission changes overrides the repeated results from statistical analysis that shows that models with more then a few input parameters cannot be justified [ref]. The point is that cost models are as much political and psychological as they are formal statistical models. A major advantage of spreadsheets is that these models are assessable to both experts and non-experts for review and comment.

Budgeting models interface with institutional rates and factors databases. Has been a major debate at JPL between various approaches. One emphasizes flexibility so can modify WBS structure and other elements quickly to support the dynamic proposal environment. The financial side of the laboratory is more concerned with passing an audit, getting
consistent results, ease of use by minimally technical people. A major issue that has arisen with these types of models is that in the early lifecycle the WBS and system components are very dynamic requiring that the tool be easily modified. This has at times lead to some significant computational errors as links get broken.

6.0 Conclusion

Conclusion will distill out the complexity characteristics we describe in the various case studies. These will help in the dialogue of what are the complexity variables and how to define them. We will also propose some ideas for documentation, review and testing of spreadsheet intensive systems.

References

>>We will add full references in final version<<


