How Spreadsheets Get Us to Mars and Beyond

Jairus Hihn
California Institute of Technology/
Jet Propulsion Laboratory
jairus.hihn@jpl.nasa.gov

Scott Lewicki
California Institute of Technology/
Jet Propulsion Laboratory
scott.lewicki@jpl.nasa.gov

Belinda Wilkinson
California Institute of Technology/
Jet Propulsion Laboratory
belinda.wilkinson@jpl.nasa.gov

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 lists, managing requirements, monitoring progress, planning budgets, developing the initial concept designs, and providing 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 share our approaches to documentation, review, and verification of these types of systems.

1. Introduction

Spreadsheets are indeed ubiquitous throughout 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\(^1\) (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 lists, 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 questions that arise are: “When should we start treating them like software?” and “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.

In this paper we will share our lessons learned in building various spreadsheet intensive systems and

---

\(^1\) The research described in this presentation was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.
applications. We will describe approaches to documentation, review, and verification of these applications and systems. These approaches are based on tailoring the standard JPL software development practices. The following spreadsheet development and/or operations case studies will be documented:

- Case Study 1: Managing Software Quality Improvement
- Case Study 2: SMART, Software Measures Archiving and Reporting Tool
- Case Study 3: Cost Estimation Models

In the last section of the paper we will propose a number of exploratory ideas as to the dimensions of spreadsheet system complexity based on our experience in developing and using these various applications. This is clearly an area that has research potential.

2. Case Study 1: 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 just 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 the need to improve 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.

2.1 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 of 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. The checklist, associated with each of the software products and projects, captured the types of documentation that were generated and the tools that were utilized on the project.

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.

### 2.2 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 [1].

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 relationship.

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.
3. Case Study 2: SMART, Software Measures Archiving and Reporting Tool

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. The response was to implement a software metrics system. The responsible group was the Measurement, Estimation and Analysis Element (MEsA) of the SQI organization. MEsA is responsible for establishing and maintaining the JPL software metrics program. As good software engineers, we wrote 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 the metrics system began to take form, one of its major interfaces was the low-spreadsheet. In fact, spreadsheets were used to address both of the concerns expressed by our potential users.

SMART is the acronym for the Software Measures Archiving and Reporting Tool, a metrics 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 used in the software development lifecycle. SMART consists of a metrics repository, a management dashboard containing charts based on an analysis of the metrics, and planning and cost models. Categories of metrics include effort, size, defects, requirements, schedule, and process. Figure 1 shows an overview of the SMART system. Early on, we decided to use institutional standard processes in our development lifecycle. In addition to the concept of operations document, we wrote a requirements document, a software management plan, and various design documents.

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, 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>Process:</th>
<th>FY02</th>
<th>FY03-06</th>
<th>FY07-09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual collection procedure of metrics data at conclusion of development of a software product</td>
<td>Formal collection procedure for metrics data collected at major milestones</td>
<td>Direct and automated collection (with use of spreadsheet) for both metrics data collected at major milestones and data collected during the development lifecycle</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Repository:</th>
<th>FY02</th>
<th>FY03-06</th>
<th>FY07-09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spreadsheet</td>
<td>Prototype database for MEsA use only</td>
<td>SMART: an online database with a Manager’s Dashboard in spreadsheet</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: SMART System Overview

Figure 2: Overview of Capabilities Development
The current list of features provided by SMART includes:

- Built using a database and workflow tool
  - Workflow, templates, and triggers
  - Different lifecycles (development and maintenance)
  - Different types of metrics (milestone and monthly)
  - Relational database
  - Import and export of metrics using spreadsheet interface
  - Data validation of spreadsheet entries
- Data organized hierarchically by program and project
  - Metrics for almost 300 pieces of software are currently in the SMART Repository
- Prototype Management Dashboard
  - Creates 12 charts, based on Monthly Metrics, in a spreadsheet that automatically updates
  - Spreadsheet provides a pushbutton to save data to SMART Repository
  - Metrics charts are part of same workbook from which data was imported and exported
  - Metrics charts support all CMMI tasks
- Automated data collection supporting
  - SLiC (code counter tool)
  - PRS (defect system)

3.1 Process and Documentation

Since SMART was developed as part of the SQI organization, we decided to follow the institutional SDSPs and also provide compliant documentation even though SMART is an infrastructure task and is very small in size (1 to 1.5 developers). The SDSPs and associated documentation are designed to be tailored, as necessary. This enabled us to scale the development processes and documentation to the size and required reliability of our system. In addition, the development of a more comprehensive set of documentation based on the SDSPs was conducted as a proof of concept to evaluate if it made sense for small IT tasks. Reviews of the documents were conducted with the appropriate stakeholders, using a combination of document walkthroughs in meetings and written feedback based on reading the document. The documents developed and reviewed included:

- Concept of Operations
- Requirements Document
- Requirements Table (spreadsheet)
- Software Management Plan
- Architecture Diagram
- Database Dictionary (spreadsheet)
- Design Document and Artifacts
- User Interface Design Document

This proved to be very effective in our environment and demonstrated the usefulness of following standard processes and producing the required documentation in a very small task.

The documentation allowed our stakeholders to have a much clearer understanding of what was to be developed and buy-in to our objectives and approach was greatly increased. It allowed the development team to understand issues that arose and to resolve them appropriately. In addition, we decided the overall design would need to be consistent with the goals of data integrity, security, extensibility, availability, and maintainability with associated priority levels, where data integrity had a priority level of one. This has driven the choice of capabilities and the design decisions throughout the development.

The required effort to do this was reasonable in size (about five work months) and we feel that other very small tasks and spreadsheet intensive development tasks should take the time to do this.

3.2 Validation Methods

As with the documentation, SMART’s development lifecycle was also based on standard JPL software practices. SMART was implemented via iterative development phases with regularly scheduled demos. During the development of SMART, the demos were integral to verifying the implementation of the requirements for the system, an activity specified by the SDSPs. In general, we have found that frequent demos are effective in obtaining stakeholder buy-in and in performing verification of user requirements for all systems with extensive user interfaces.

In addition, as the use of spreadsheets obviously involves manipulating data it was necessary to identify formal data verification activities and capabilities. These verification activities included:

- Standard procedures for the processing of data submitted by a software task were written by MEsA. The procedures require several steps depending on the type of metrics involved. If MEsA has to ‘clean the data’ and revise the metrics then the revised metrics are reviewed with the contributing software task. Also, at least two members of the MEsA team must perform a quality check of the data before the submitted metrics can be considered to be in a ‘completed state’. Once the contributed metrics reach the ‘completed state’, only a member of MEsA can make any further changes to the data.

---

2 There is an externally released version of the Managers Dashboard that can be obtained from the Authors.
• All data fields in the database have access control lists so an unauthorized person cannot accidentally alter the data. The extension of this feature to the data in the spreadsheet has not been done. See further discussion in 3.7
• A complete history log is maintained in the database which identifies when a data item has changed and who made the change. The extension of this feature to the data in the spreadsheet has not been done. See further discussion in 3.7
• An extensive quality check was also performed on data from old spreadsheets and a previous prototype database before importing the data into the SMART Repository by collecting all of the data in a large spreadsheet and rigorously checking the data for correctness and uniformity. This step took two people about a month to complete. Roles alternated so that a ‘fresh-set-of-eyes’ performed the verification step for correctness and completeness.

3.3 Spreadsheets as the Human Interface

Because our users are so comfortable with using spreadsheets, the decision was made to provide a special SMART-format spreadsheet for the entry of Monthly Metrics. The metrics can then be uploaded to the SMART Repository via a pushbutton interface. A Management Dashboard is automatically updated based upon any new metrics entered. There is a hidden worksheet containing the computations to create the charts on the Management Dashboard.

Since it is so easy to change data in a spreadsheet we needed some level of configuration management. At one point, we had a user who changed data that had been entered several months previously. To prevent this, the spreadsheet was locked and pushbuttons were provided in the spreadsheet for the following functions:
• Add a new spreadsheet column to enter the current month’s metrics. All columns for previous month’s metrics are locked and cannot be unlocked by the users although MEsA members can unlock the columns, if necessary.
• Submit the current month’s metrics to the SMART Repository. Once submitted the entry column is locked.
• Unlock the current month’s entry column to allow the user to update the metrics for the current month.

3.4 Using Spreadsheets to Simplify Interface Complexity with Institutional Tools

At JPL institutional tools are available for activities such as requirements management, defect tracking, and counting source lines of code. Creating an interface between SMART and these tools was problematic. The solution was to export data from the tools into a holding area within SMART, process the data, create a SMART-format spreadsheet from the data, and then upload the spreadsheet to the SMART repository.

3.5 Future Direction: Spreadsheets as a Software Design and Implementation ‘Data Structure’

The metrics program continues to evolve and we recently realized that we could implement the capability to use a spreadsheet containing the set of collection metrics as a control document or ‘data structure’ to automatically build a data-driven database schema and the user interface forms³. The enhanced metrics set has been defined in a configuration managed spreadsheet which includes the following data items for each metric: metric name, display label name, measurement category, as well as attributes such as applicable task size (small, medium, large) and task type (development or maintenance), data type (text, integer, enumeration list, etc.), collection frequency (milestone or periodic), collection source, collection responsibility, storage location, and reporting details. Other than updating the set of collection metrics in the spreadsheet, the only planned manual action will be to use a forms manager to define the layout of the data in the form. This will result in a more robust metrics system that will be easier to maintain.

3.6 Major Strengths

To summarize, there were a number of major contributions to the metrics system from the use of spreadsheets. They included the following:
• Our users are experienced with using spreadsheets and spreadsheets are readily assessable. This user acceptance reduced the need for training.
• The use of spreadsheets allowed the import and export of data between humans and systems and also between two systems. This solved the

³ The idea is by Carlos Balacuit, member of the SMART development team.
problem of interfacing SMART with other institutional tools.

- Spreadsheets were an effective tool in the evolutionary design of the user interface. They allowed us to interact with our users and make changes in real time. This was especially useful in the development of metrics charts.
- The spreadsheets submitted by the software tasks were successfully used to meet the metrics related goals that were part of the CMMI Level 3 assessment obtained in 2007.

3.7 Major Issues

There also were major problems that we found by using spreadsheets. They were:

- The configuration management of the data was problematic with the spreadsheet but the use of the spreadsheet pushbuttons provided an acceptable resolution.
- There was no guarantee that the user would click the pushbutton to upload the metrics to the SMART Repository. For the CMMI tasks, MEsA performed a manual check to verify the upload. However, this is not an acceptable long term solution.
- Access control lists for the data and the history log are very important features for a metrics repository. Reasonably similar capabilities can be implemented in a spreadsheet using scripts and locking features but configuration management and performance are concerns. For example, a good solution as to where to keep the access control lists and how to control the updating of them is not readily apparent.

4. Case Study 3: Cost Estimation Models

Virtually all cost models at JPL are built in spreadsheets [2, 3, 4]. Examples of the types of software cost models developed and used at JPL are described below. Those to be discussed include the Software Cost Analysis Tool (SCAT), the Flight Software Cost Model, and the JPL Space Mission subsystem level grass roots models. A key consideration in developing cost models is the activities performed to verify and track accuracy.

SCAT (Software Cost Analysis Tool) is a Monte Carlo version of COCOMO [5] implemented as a multi-sheet model that can import from a separate Monte Carlo sizing tool. Estimation accuracy is validated and documented⁴. The results of the initial validation are published in [6]. Validation is performed by calculating the percentage of estimates within +/- 30% of actual historical data. The history of model based proposal estimates is also tracked and compared to the final development costs as they become available⁵. Usability and user error are not formally assessed. 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.

The Flight Software Cost Model is the primary cost model for estimating mission-critical robotic space mission’s software. It can run stand alone or integrated into the Team X tool set. Team X is JPL’s concurrent engineering design team responsible for early designs and estimates. The Team X tool set is a networked spreadsheet intensive system with real time parameter updating. A detailed description of the Team X tool environment is described in a paper also being presented at HICSS 42 [7].

The Flight Software Cost Model is a complex multi-sheet model that takes high level system descriptors (pointing accuracy, number of instruments, etc.) and then executes two sub-models which estimate the system size and the effort multipliers. These provide the inputs into a COCOMO based model (point estimate). The COCOMO portion of the model has the same parameter calibrations as SCAT. In the Team X environment, as mentioned above, it is part of over twenty integrated multiple workbooks that pass parameters over the network. This model was rigorously verified with formal documentation. Documentation consisted of a user guide, a model description document and validation results. The validation consisted of model performance against actual historical data similar to SCAT. The cost model was validated against several in-house, mixed development, and out-of-house missions of various sizes. The missions chosen for validation were those that have launched and had actual historical costs easily available.

Because the Flight Software Cost Model is part of the Team X spreadsheet based tool set, it was required that an additional validation step to test potential user error be performed. In a blind test, two different estimators had to produce estimates within 10% of each other using the same high level mission specification.

⁴ There is an externally released version of SCAT that can be obtained from the Authors.

⁵ For an earlier version of SCAT based on COCOMO 81, estimates from1989 through 1995 for ground software were high by an average of 3% compared to 40% under allocation of budgets from the preliminary design review.
The last models we will discuss are the JPL Space Mission subsystem level grass roots models. Each JPL organizational section\(^6\) that builds a major component of a space mission has its own spreadsheet based cost model. 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 the same manner as the Flight Software Cost Model. However, 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. An important advantage of having our cost models in spreadsheets is that anyone can pick them up and understand how the costs are derived. This makes them easy to review and facilitates discussions between the cost engineers and domain specialist engineers.

Finally, spreadsheets have enabled an important dialogue between the cost engineers and the rest of the engineering community. It is well documented that cost models are driven by a small number of factors and that most design changes have no or imperceptible cost impacts \citep{8}. The domain engineers prefer very detailed models so they can make small changes in the design parameters and see a change in their estimates. We have found that, organizationally, the need to reason about the cost of different design changes overrides the repeated results from statistical analysis that shows that models with more than a few input parameters cannot be justified \citep{9}. The point is that cost models also have political and psychological requirements that must be imposed on the formal statistical models. A major advantage of spreadsheets is that these models are accessible to both experts and non-experts for review, comment and defense of their proposals.

5. Spreadsheet Complexity

Based on over fifteen years of implementing various tools in spreadsheets, we have observed practices that increase complexity and reduce spreadsheet quality. While there is an extensive body of literature that explores software complexity \citep{10, 11, 12}, spreadsheets provide some unique considerations at least with how they combine various system features \citep{13}. Our current perspective, while it overlaps some with \citep{13}, is based on a developer’s perspective. We propose that spreadsheet complexity be divided into computational complexity and interface complexity.

Other authors may disagree and we hope this will lead to an interesting dialogue.

Computational Complexity
- Straightforward cell based computations vs. extensive use of macros
- References across multiple spreadsheets (closely related to model to model interfaces)
- 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. This is a lot like direct memory referencing in a software program which is definitely not a best practice

In response we have evolved some common practices:
- Only use macros for generating reports. Macros are difficult to debug and when combined with multiple open workbooks create serious performance issues. This is a major issue with Monte Carlo tools.
- Do not use deeply nested simple formulas. Complicated formulas in a single cell are easier to understand and maintain.
- Name variables and worksheets and never reference cells. Define a naming convention that assists with comprehension. For example, we name all sheets, tables and data items. A variable name is then built by concatenating the sheet, table, and item names.

6. Conclusion

In this paper we have described how spreadsheets have been used as an effective interface tool. A major advantage of spreadsheets as a human interface is that everyone is very comfortable with them, which greatly reduces resistance to the infusion of new tools and methods. We have also documented how we applied standard software development practices and documentation in the development of various spreadsheet applications. This has been very effective in the JPL environment and our experience should extend to other engineering organizations.

---

\(^6\) JPL is a matrix organization. A section has from 100 to 200 people with similar domain expertise, who are assigned to our various missions. For example, there is a ‘flight software and data systems’ section as well as a ‘ground data systems’ section.
We especially recommend the maintaining of a formal requirements list, documenting operations scenarios, using a demo based lifecycle with 4 to 6 weeks between builds, and writing unit test scripts to catch spreadsheet computational and procedural errors. Finally, depending on the required reliability of ones application, the use of double blind user execution tests to identify potential user errors was very effective and also was very powerful in obtaining stakeholder acceptance.

References


