Evolution of the Scope and Capabilities of Uplink Support Software for Mars Surface Operations

Marc Pack[1] and Sharon Laubach[2]
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA

In January of 2004 both of the Mars Exploration Rover spacecraft landed safely, initiating daily surface operations at the Jet Propulsion Laboratory for what was anticipated to be approximately three months of mobile exploration. The longevity of this mission, still ongoing after ten years, has provided not only a tremendous return of scientific data but also the opportunity to refine and improve the methodology by which robotic Mars surface missions are commanded. Since the landing of the Mars Science Laboratory spacecraft in August of 2012, this methodology has been successfully applied to operate a Martian rover which is both similar to, and quite different from, its predecessors. For MER and MSL, daily uplink operations can be most broadly viewed as converting the combined interests of both the science and engineering teams into a spacecraft-safe set of transmittable command files. In order to accomplish these ends a discrete set of mission-critical software tools were developed which not only allowed for conformation to established JPL standards and practices but also enabled innovative technologies specific to each mission. Although these primary programs provided the requisite capabilities for meeting the high-level goals of each distinct phase of the uplink process, there was little in the way of secondary software to support the smooth flow of data from one phase to the next. In order to address this shortcoming a suite of small software tools was developed to aid in phase transitions, as well as to automate some of the more laborious and error-prone aspects of uplink operations. This paper describes the evolution of this software suite, from its initial attempts to merely shorten the duration of the operator's shift, to its current role as an indispensable tool enforcing workflow of the uplink operations process and agilely responding to the new and unexpected challenges of missions which can, and have, lasted many years longer than originally anticipated.

I. Introduction

Mere months before the first of two spacefaring rovers was scheduled to land on Mars, members of the tactical operations team for the Mars Exploration Rover (MER) mission had a problem. Their ambitious plan to command the spacecraft every day was meeting up against the harsh reality that trial runs of the uplink process were taking hours longer than what had been allocated. Without significant improvements, a goodly portion of the mission's precious 90 planning days would be lost.

The uplink process is composed of a logical sequence of phases, each more concrete than the last. And each phase is separated from the next by a different review meeting, wherein the results of that phase's primary software tool (plus ancilliary reports) are examined for various levels of feasibility and correctness. The breakdown of the original uplink process at that point in time was as follows:

1) **Science Phase**: the different science teams meet to discuss their plans for the day. The primary software tool is the Science Activity Planner (SAP) program. Operators use the tool to input science activities, and after each team makes refinements to the activities they own (with difficulty since the tool does not have a branch-and-merge capability), the final merged plan is reviewed and saved out in a proprietary XML format.
2) **Planning Phase**: the Tactical Activity Planner (TAP) takes the science and engineering teams' interests and combines them into a time-ordered activity plan using the Mixed-initiative Activity Plan Generator

[1] MSL Lead Sequence Integration Engineer, Planning & Execution Systems Section, MS 264-623
[2] Supervisor, Planning & Execution Systems Engineering Group, MS 301-260

(MAPGEN) program. Input and output files are in a proprietary MAPGEN-specific format, and several rudimentary reports are created which describe conflict checks and resource usage.

3) **Sequencing Phase**: all the uplink teams transform their portion of the approved activity plan into sequences of commands using the Rover Sequence Editor (RoSE) program. Operators type commands into the GUI, which are later output in a proprietary XML format known as Rover Markup Language (RML). The sequences themselves are reviewed in RoSE (primarily the backbone sequences, which kickoff subsidiary sequences at particular times and/or in particular orders).
4) **Validation Phase**: the first-shift Sequence Integration Engineer (SIE) gathers all the sequence RML files and converts them into a proprietary input format for the Sequence Generation (SEQGEN) simulation program. Output of the tool is a proprietary-format Predicted Events File (PEF), from which some reports of limited utility are derived.
5) **Compilation Phase**: the second-shift SIE gathers all the sequence RML files and converts them into a proprietary input format for the sequence translation (seq_translate) compiler program. Output is a radiatable file containing the binary equivalent of the text commands, without any accompanying reports.

The descriptions above, while accurate summarizations, are greatly simplified (for more detail refer to Mishkin, et al. [1]). Even so, a casual reading of them lays bare the tremendous hurdles which would need to be overcome: many of the powerful primary programs required significant manual effort in order to be best utilized; the few reports available provided only a fraction of the necessary understanding of the phases' output, and in a form which was extremely difficult to digest; furthermore (and possibly worst of all) not a single output from any phase's primary program matched the input format of the subsequent program.

II. Everywhere: A Very Good Place To Start

It's impossible to overemphasize the importance that systems engineering played at this stage of the mission. With each part of every phase of the process needing enhancement, and with the resources for that enhancement limited to less than two people, the lack of skilled oversight could have been disastrous. Attention might have been focused upon only one of the phases, or a simplistic beginning-to-end (or even end-to-beginning) approach could have been adopted. Instead, the perspective encompassed the entire uplink process and programming was aimed wherever it was deemed to be needed the most.

One might think that the obvious first effort would center on the compilation phase; after all, it's worthless to worry about the preceding phases if you can't command the rover. But JPL has an overriding dictum that is drummed into everyone from the earliest career hires to the most senior management: protect the health of the spacecraft. It is considered perfectly acceptable, even if the entire uplink process has progressed without incident and the files are moments away from being radiated, to abort the commanding for that day if doubts have arisen regarding how those commands might "break" the vehicle. And given that overriding institutional philosophy, the first phase to receive attention was obvious: phase four, validation of the command load.

MER chose to accomplish command validation with the venerable tool SEQGEN, an institutional multi-mission program which for many years has provided missions with an extremely powerful (down to the thousandth of a second) command/sequence-of-commands simulator. In addition to verification of individual commands SEQGEN provides a proprietary language which allows each mission to adapt the program's operation in such a way as to allow it to check how commands interact with each other, as well as against a representative state of the spacecraft itself. Perhaps due to SEQGEN's power, however, the results of a simulation run can be quite daunting. A single Predicted Events File (PEF) can contain thousands of time-stamped lines (logical lines, which can be wrapped many times the maximum length of a physical line) representing every event (command starts and ends, state changes to any of thousands of variables, errors/warnings/notations, etc.) that occurs within the SEQGEN engine.

It's no surprise, given this, that the very first piece of software written for the Command Uplink Generation and Review (CoUGAR) tool suite was designed to ingest a PEF and then provide a readable report of all the errors and warnings contained therein. An additional report detailing the start and end times of every sequence was also created by this first tool, and that report ended up being the input for the second CoUGAR tool, which used the information to enhance a graphical view of the simulation. Thus was born a pattern which would be replicated many times over in the toolset: create data at one point in the process that is not only useful in and of itself but can also be utilized to enhance later parts of the process.

Given the limited resources available for design and implemention of the tools, it is noteworthy that much of the work was accomplished by adopting an agile development paradigm whereby both people programmed side-by-side. Of even more importance was the designed-from-the-start ability to be able to quickly respond to problems or insufficiencies with the tools, while maintaining best configuration management practices appropriate for the tools'

class of software. Testing was also made relatively easy, as the entire suite was designed to maintain an awareness of its environment such that it could sense when it was being run on a non-operations workstation or outside the official operations directory structure, and therefore utilize the sandboxed resources allocated specifically for development efforts (and thus prevent the corruption of genuine flight data). While other programs required days (at best) or weeks to get changes installed, requisite modifications to the CoUGAR tools could be ready within minutes and immediately used during tactical operations.

III. Software! What Is It Good For?

At its worst, software can be terribly frustrating. There are well-known examples of programs that promise the moon, but require herculean effort to extract but a pebble. And that assumes that the program functions correctly, or that you could even reliably install it on your computer.

But at its best, software can be incredibly enabling, allowing people (in groups of one or more) to accomplish things in short periods of time which would have been nearly impossible to even attempt by themselves non-digitally. Each MER phase's primary software program, whether strictly text-oriented like SEQGEN or with a newer Java GUI like RoSE, are capable of handling hundreds (if not thousands) of different kinds of inputs while performing tremendously complex operations and communicating back to its human operators. That kind of power comes from marshalling a huge amount of personnel to design, implement, validate and install these programs, an effort which also commands the lion's share of management's attention. And while it would be inconceivable to attempt to mount a space mission without these mighty oaks in the software forest, it's easy to lose sight of the shrub-and-moss tools which provide the most basic, fundamental service that software can provide.

That simple service can best be described as: perform a predictable task faster, and with fewer errors, than a human being could perform that same task. On innumerable occasions, particularly at the outset of the development of the CoUGAR tool suite, this straightforward tenet provided for significant savings in the time spent attempting to plan a martian day (known as a "sol"). To use our previous example, it certainly would be *possible* for someone to scroll through tens of thousands of lines of a SEQGEN-output PEF looking for the words "ERROR" or "WARNING" (which may be split by carriage returns), but that would take hours and risk that person's eyeballs missing their target. By far a better solution is a simple script that de-wraps the physical lines and then easily finds and writes just the errors and warnings to a separate report.

A stronger example can be found at the beginning of the simulation phase: running SEQGEN. As previously noted, the simulator takes many different types of inputs: start & stop times, configuration files, spacecraft states, MER-specific adaptations, output specifications, and of course all the commands from the current plan (which must be converted from RML to a legacy format). It is typical for dozens of inputs to comprise a single run, which eliminates the feasibility of specifying them all as command-line arguments. Instead a specially-formatted file (with the extension .ENV) lists all the above information (and more). Initially the SIE was given a template .ENV file to bring up in a text editor so that he could change the sol-specific data and add a list of command files. It was time-consuming drudgery and the only validation tool was SEQGEN itself (which might take anywhere from a few seconds to several minutes to complete, after which the user needed to examine a log file to determine if something went wrong).

An early CoUGAR tool replaced the manual process, automating the creation of the .ENV by combining the unchanging lines of the template with lines specific to a particular sol. The local directory is examined to add in all the command files, and the SIE is prompted for unknown information such as the start and end times of the simulation. A complete .ENV is written out in a matter of seconds, and the duration of the uplink process is concurrently reduced by a score of minutes.

Although the time savings for just this one tool is substantial, of even greater significance is where the tool is located in the uplink process timeline. Because it provides aid in progressing from sequencing to validation, this tool represents a vital class of "glueware" which links together the outputs and inputs of contiguous phases. And this tool wasn't alone. During the course of the prime mission the CoUGAR suite constructed bridges over every transitional gap, essentially converting each phase from an isolated island to a roadstop on the uplink highway.

IV. Son Of A Diddly!

Once some of the CoUGAR tools were put into practice a disturbing new trend was detected: users were making mistakes. Of course, slipups can happen even under the best of circumstances. But MER was a mission that was going to operate for several months on a Martian work schedule, which would result in people coming and going at all hours of the day or night. A great deal of research was put into mitigating the deleterious effects of Mars time,

but there was no avoiding the fact that some level of sleep deprivation would occur and the consequent loss of acuity might translate to a greater propensity for error.

The first line of defense against user error is also the most obvious: syntax checking. If the software prompts a user to answer a yes-or-no question and then forks one way for "Yes" and a different way for "No" then it needs to ensure that the user is not allowed to answer the question with a mistaken "eYs" or a contumacious "Maybe". Even if the syntax is correct the user's entry must also match the intent of the input. Requesting a time of day could result in a response of "12:30:00", but the response "65:43:21" passes the same syntax test while failing as a valid time.

As is well understood in the world of software, the further upstream you catch a problem the easier it is to fix. Building error-checking into the tools was no different. Consider our popular example of running SEQGEN. When creating the .ENV file, the SIE is prompted to enter the start time. Using an unexamined string could result in instructing SEQGEN to start at an invalid time, a fact that would not get discovered until after the program exits and the SIE has examined the log file for errors. But catching (and rejecting) a malformed time string at the entry point saves both time and aggravation.

But wait, there's more! Recall that the SIE is actually being prompted for both the start and end times of the SEQGEN run, which adds an additional dimension to the checking. Now both entries are checked for correctness individually, as well as being compared to ensure that the start time occurs before the end time. Other checks at this point include confirming that the start time is not in the past and that the end time is not too far beyond the start time. This may seem to be an impressive level of foresight on the part of the developers, but regrettably most of the error checking has actually been in response to genuine mistakes.

A final layer of error prevention that the CoUGAR tools came to deploy involved the prospect of bypassing typed input altogether. Whenever possible, default values are offered up as the response for simply pressing the <Return> key, and since these values are internally-generated strings they have little chance of being incorrect. To rely on our example one more time, it was determined that SEQGEN runs would typically start at 8:00 in the Martian morning, and end at 12:00 on the next Martian day. Providing these two times as the defaults prevents the possibility of error on most days since the SIE can simply hit <Return>, but still allows for nonstandard times (and the error-checking of same) should the situation require it.

It is, of course, impossible to anticipate all conceivable erroneous contingencies, which is why the tool suite eventually came to have built-in capabilities for forensic analysis. Whenever appropriate, tools would save important information to log files as they were running. Even more useful, it turned out, was insisting on having the operators use a fixed arrangement of xterm windows, each of which had logging turned on. Many times over the years on MER, an (otherwise untraceable) error was located, understood and (of course) prevented from ever happening again thanks to being able to scrutinize the terminal logs. But under the most unusual and difficult-to-trace circumstances, the tools have been left with only one option: phone home. There are key areas in the tool suite where captured failures are not only communicated to the user but also emailed back to the CoUGAR developers with all the requisite information necessary to understand not just what went wrong but how to respond to it going forward.

V. Power Begets Power

As the reach of the tools expanded throughout the process, they began to establish a paradigm of power and convenience which resulted in an unexpected outcome: innovation. The entire uplink process had been documented, and all the requisite steps (while lengthy and inconvenient) were well-understood. Development of the tools could have simply followed the established path, but (thanks to the involvement of systems engineers) new opportunities were gleaned as improvements progressed.

One example of that is sequence management. On a typical MER planning day, dozens of sequences are in use. Some are brand new and need to be radiated to the spacecraft, while others are already onboard and are simply being reused that day. And of the sequences which are getting radiated, some will replace older onboard versions while others are only meant to be used that one day and should be deleted the next. Initially these (and other) aspects were being communicated informally by each sequence's owner, but not tracked or organized in any official manner. However as the level of automation for the uplink process continued to mature, it was realized that a hitherto-unforeseen product, the Sequence Management Table (SMT), was required.

The SMT became the central repository of each day's sequence information, and as such was a locus for catching many different kinds of problems. Such issues included accidentally thinking a sequence was already onboard when in fact it needed to be radiated, or preventing the deletion of a sequence which was still in use. It also provided valuable information to many tools downstream in the uplink process, including those which compiled the

sequences and those which created sequences for the next plan. In fact it ended up being so central to uplink that it became a reviewed product itself!

As previously mentioned, one of the most vital roles filled by the CoUGAR tool suite was in closing the gaps that existed between the phases. The largest of those gaps was in the transition from activities to sequences. The SIE initially was required to load a generic template into the sequence editor, and then to adjust all the times to create the backbone sequence for the sol being planned. Additional typing was required to create the subsequences that would run the science commands as they had been laid out in the activity plan. To address this laborious hindrance, a preliminary tool was written to extract an ordered list of commands from the activity plan, while a subsequent tool converted that list into a spreadsheet which could be edited to assign science commands to the subsequence which calls them. Next a complex utility was written to merge the edited spreadsheet with the backbone template, and a rich set of spreadsheet annotations was created to aid in adjusting the backbone sequence without further intervention. And after patterns related to the final spreadsheet edits began to emerge, the programs were further enhanced to pre-edit and pre-annotate the spreadsheet and reduce the level of manual intervention to virtually nothing. This spreadsheet concept, invented out of necessity and refined through experience, was key in overcoming the largest hurdle of phase transition.

A different kind of transition was also enabled by the tool suite: plan-to-plan. A variety of different pieces of information are managed by the tools for planning on subsequent days, including (but not limited to) the simulated spacecraft state, a predicted list of onboard sequences, future sequence deletions, and SEQGEN commands from the end of the plan. Other tools will search backwards in earlier plans to gather previously-modeled data files for use in the current day.

Reviewed products and reports were another hotbed of innovation for the CoUGAR tools, which created a variety of files that presented information in ways that (while seemingly obvious now) were completely unheard of initially. One noteworthy report began as the PEF-derived simple table noted above, where each sequence was listed with its start and end times. A separate tool created to aid the sequencing phase generated a "call tree" of the sequences, where sequences which called sequences which called other sequences (etc.) were represented by an indented cascade of sequence names. Recognizing the visual value of the cascaded format, a new report was derived from the PEF which showed the same cascaded tree from SEQGEN's perspective and included the timing of each sequence.

But the true lightbulb-over-the-head moment came when a new CoUGAR tool combined both of the call trees into an annotated tree containing everything that SEQGEN knows (which sequence calls which, along with when it starts and ends) with everything SEQGEN didn't know (additional sequence calls that may have gotten cut off in the simulation, useful command comments that get lost in translation). The resulting report was a multi-dimensional display of the entire plan which could easily highlight areas that may not have had enough time to run and which allowed the reviewers a succinct (yet informative) overview of what was happening that day. The cascaded tree report was so useful that other teams built on it to report their own results.

VI. We Control The Vertical

In the beginning the CoUGAR tools would dot the landscape of the uplink process as they were written, small tools that filled the most immediate needs where determined by the system engineers. But as the tools proliferated, they began to occur in clusters around particularly knotty areas of a phase. As described above, the first and second tools were scripts which derived useful reports from SEQGEN's output PEF. A later tool was created to filter yet another PEF report to improve that report's signal-to-noise ratio. And a fourth tool gathered all these outputs and published them to an official reports destination directory.

The invocation for each of these tools often included the particular sol that was being planned, and in some cases the order in which the tools were run was important. Naturally, the best way to handle this situation was with another tool! This new script lived "above" the others conceptually, a superscript which not only saved time and errors (software's bread and butter, after all) but also began (on a small scale) to create a workflow of sorts that herded the necessary data in the right direction for review by the team. Each time a new cluster of simple tools would appear, a superscript got written to invoke them. And as the process became more and more automated, the superscripts themselves were subsumed into grander superscripts above them, and so on. And even though the final goal may have been to have a single tool accomplish most (if not all!) of an entire uplink phase, the tools were always designed with an "escape clause": should a superscript fail for any reason, or should an anomaly or other event disrupt operations to an extent not covered by the superscript, then each and every lower-level script would be maintained to assure that the operator could still get the job done with as much CoUGAR help as still feasible.

To expand on our previous example, our highest-level simulation superscript would eventually:

1) Confirm that the command RML had been translated to SEQGEN's legacy format (and abort otherwise).
2) Create and populate a new (possibly incremented if there were multiple runs) subdirectory containing all the inputs for SEQGEN.
3) Create a .ENV file based on user inputs and the contents of the directory.
4) Run SEQGEN with that .ENV file.
5) Alert the user if SEQGEN did not complete successfully.
6) Otherwise create and publish a variety of reports.
7) Copy the SEQGEN-created final spacecraft state file out to use as input for the next planning day.

As the superscripts grew in power and cooperation, they provided a new opportunity to inhibit user error: workflow enforcement. Inside the superscripts commands were guaranteed to be run in the correct order, but users might accidentally skip a step or run commands in alternative orders without understanding the underlying interdependencies of the data flow. And occasionally, workflow enforcement needed to be programmed in response to actual tactical errors.

As previously noted, the uplink process was documented, both internally and externally. "Internal" documentation refers to the fact that each and every tool in the suite was built to accept a "-h" option, which would print out a usage statement for the tool summarizing its functions and completely describing all options and arguments. A regularly-updated (by another tool) README file (both text and HTML) collected the usage statements of every member of the CoUGAR suite for ease of reference. For external documentation, the SIE's had a written procedure that they dutifully followed every day, typing in the commands and inputs as described therein. This living document was perhaps the greatest victim of the success of the CoUGAR tools, for as each step became easier the procedure shrunk accordingly. Eventually the body of the procedure was dwarfed by the appendix of "retired" steps, and a side project was initiated to automagically create a customized HTML "checklist" containing what few commands and instructions were still required to plan for that day.

VII. Piled Lower And Deeper

As new tools were created throughout the process, patterns began to emerge whereby fundamental lower-level concepts and operations would get used repeatedly in different areas. Best practices dictated that the commonalities be centralized, not just in order to avoid code duplication but also for ease of maintainability should external changes occur. For instance, the web site which contained the daily journals of the different teams was accessed multiple times throughout the day by the CoUGAR tools, for both reading and storing information and reports. The URL for this web site has changed several times throughout the mission, but because the value is only recorded in one place (a script which serves up this and other mission "service" locations), it was trivial to update the entire tool suite.

Another file served as a library of commonly-used variables and functions for the many shell scripts in the suite. By sourcing this one lower-level file near the beginning of a script, values derived from the user's shell environment (today's sol, number of sols in the plan, the path to the reporting directory, etc.) became instantly available and provided an excellent launchpad for brand new scripts to get their bearings. The functions available through this same file (select from a list of files matching a wildcard, make a choice from a fixed list of options or use the default, calculate the rolled-over sequence sol from the true sol, etc.) were so basic as to be used countless times across every phase of the process.

An unsettling phenomenon began to occur soon after many of the superscripts got installed. Since these tools were invoked as commands in a terminal, the primary method for communicating back to the user was (obviously) printing messages to the screen. But even though great care was taken to format the messages in order to present normal outcomes differently from abnormal ones, occasionally a warning or error was missed by the operator. In order to counter that issue a new utility was created which colorized output. Benign output was white (on a black background), warnings were shown in yellow and errors in red. The severest errors forced the user to take note because a window containing the message would pop up and need to be eliminated manually before the process continued. Use of this new subtool wasn't mandated to replace all the print statements in the entire CoUGAR suite, but it allowed for targeted emphasis of problems where that emphasis was deemed most necessary.

Possibly the most powerful tool in the suite was originally created as a low-level service to meet one of the most pervasive needs of the uplink process: time manipulation. Because the mission was solar-powered and did most of its work during the day, planning tended to think in terms of the Mars local mean solar time (LMST) when scheduling tasks for the day (start driving at 11:00 LMST; take some pictures at 12:30 LMST; go to sleep at 14:00 LMST). But major programs like SEQGEN, which is designed for all classes of missions (many of which are not

clearly tied to a planetary clock), think only in terms of what time it is on our planet (the start of surface operations is 2004-003T13:36:14.814 UTC for example). Leaning once again upon our SEQGEN example, recall that the SIE is requested to specify the start and end times of the SEQGEN simulation. From the rover operations perspective, the start is most often 08:00 LMST and the end is noon LMST of the following sol. Originally the SIE would be required to manually enter the LMST time into a time-converting web site, and then transcribe the UTC result in order to create the .ENV file for SEQGEN. The CoUGAR time tool originally provided a simple (and scriptable) method for converting between the two time systems, but as the suite of tools grew the demands on time conversion grew as well.

It became necessary to support new time systems (such as the spacecraft clock, which keeps a monotonically increasing count of seconds) as well as the deceptively difficult operation of "time math". Oftentimes, a script would need to add or subtract a certain number of seconds or minutes from an user-entered time. While it's easy for a program to reduce 2005-001T12:34:56 by five seconds (only the seconds field is modified), that same program must also be able to subtract five seconds from 2005-001T00:00:00 (every single field is now different!). As more and more time-conversion functionality became required, the tool eventually supported several more time systems, the manipulation of basic durations (both earth and mars) and even one-way-light-time (how long it takes to radiate a command to Mars, based on the respective positions in their orbits of both planets).

VIII. Like Sands Through The Hourglass

As previously noted, MER's prime mission was designed to last for ninety days. The entire project intended to work Martian time for the duration, and the CoUGAR tools were designed to reflect that simplifying assumption. At the start of surface operations, the tools had already proven their worth. With the benefit of operational readiness tests as a proving ground the size of the suite stood at over seventy programs, and it had shaved enough hours off the clock to allow daily operations to commence. But by the end of those three months, the initial excitement of operating a rover on another planet was being subsumed by the miserable hardship of living independently of Earth sunshine. Furthermore, the rovers were proving themselves surprisingly long-lived. Management made the decision to transition to an Earth-time-based schedule, the first step of which was to squeeze down the duration of the process to fit into a more conventional daytime workshift. This was by no means a trivial task, as it would require a reduction of roughly seven hours for the uplink process. Besides the time savings already realized by the growing CoUGAR tool suite, new tools were brought to bear on the issue, recovering valuable time (and every minute counted) through progressively more sophisticated automation. Even more significantly, systems engineering analysis identified a labor-intensive task which was being performed separately by different teams, and newly-minted tools created a first-of-its-kind (for MER) cross-team data flow which completely eliminated the time sink. But as the calendar pages continued to turn, and the rovers stubbornly maintained their health, it became clear that a major paradigm shift in operations scheduling would be necessary: the normal work week (for more information on the extended mission refer to Mishkin, et al. [2]).

All the tools had been designed with the idea that the team would plan one sol at a time, and then come back the following day and do it all again. But giving the team weekends and holidays off meant that the suite would now need to support both standard single-sol operations as well as multi-sol plans. This required major changes not just to the simple tools but all the way up to the highest superscripts, with particular attention paid to increased enforcement of the workflow (to prevent, for instance, work to begin on sol 3 before sol 2 was completed).

The rover's longevity created challenges in many other areas where limited assumptions had been made. For instance, the daily backbone sequence that the SIE creates is named to match the martian sol it belongs to. But that naming convention (constrained by the spacecraft software) only provided enough space for a little more than four hundred sols, and it was thus decided that the sequence name would roll over. The tools could no longer simply use the current sol, now they had to derive a modulus-400 value from the current sol. But that problem paled in comparison to the project-wide "Sol 1000" issue, since all the software had been designed with a seemingly-valid (at the time) assumption of needing no more than three digits for the sol. In a miniature repeat of the global "Y2K" problem, sizable resources were applied to finding, changing and testing all of the project's software (the CoUGAR tools, being mostly scripts, were fairly easy to examine) to assure a smooth transition to 4-digit-sol operations.

External factors would exert their influence on the mission as the rovers continued to endure. Budget reductions were met with the merging of some positions, thanks to the ever-increasing automation brought about by the CoUGAR tools. For the remaining positions, existing personnel eventually moved on and their replacements arrived without the depth of knowledge that came from experience in the early days of the mission. The lack of familiarity that new staff members suffered from often revealed never-before-seen vulnerabilities in both the process and the tools, necessitating software changes that were usually, but not always, quick and minor.

In response to one alarming anomaly, the process was changed (with CoUGAR support) to prevent the SIE from being able to progress too far without reaching a "cognitive gateway" during a review. A word was chosen at random from a dictionary and placed at the bottom of a report. This "word of the day" was invisible on the SIE's workstation, but could be seen on the reviewer's screen. The superscript being used would pause at this gateway point and wait for the SIE to correctly type in the word, as spelled out by the reviewer, before continuing. There is probably no harsher example of workflow enforcement in the tool suite, but there has also been no reoccurrence of the anomaly either!

After a different incident, multiple tools were created to close the loop between what had been sequenced, what had been reviewed, and what was prepared for radiation. Although each of these three products is essentially composed of the same spacecraft commands, their formats are very different and comparisons are not straightforward. Nevertheless, one of the highlights of the daily shift is observing a pair of "none"s in the comparison reports (signifying no differences in content nor in intent), as it indicates successful operation of the process (and that the shift is nearly over). Both the word-of-the-day and the "pair of nones/nuns" became a daily ritual that the team actively looked forward to, helping to lighten the atmosphere during intense operations in the uplink room.

The tools' ability to agilely respond to changing conditions was a blessing indeed, but at the same time it was also a curse. Had the hurdles to modifying code been higher, the suite's can-do attitude might have been tempered by a greater sense of finality. As it was, however, a running joke between members of the development team was that the CoUGAR tools were 90% complete. They were 90% complete after six months of operations, 90% complete after five years (and hundreds of additional modifications), and they remain 90% complete to this day.

IX. Deja Vu All Over Again

By the time the Mars Science Laboratory (MSL) landed on Mars, a brand new suite of CoUGAR tools, tuned for the new mission, was already up and running. One might think that a significant amount of inheritance would be possible, given that both MER and MSL were six-wheeled robotic geologists on Mars and that the same five phases comprised the uplink process. But the reality of the situation is that many of the details underlying the phases has changed. The primary software programs for the first three phases have been completely replaced by a single large integrated program, and this integrated program is also able to invoke the primary legacy programs from the last two phases (which are updated versions of the MER counterparts). The file system layout that the CoUGAR tools rely on changed too much to directly port tools from MER to MSL, as did some of the supporting services such as the previously-mentioned journaling web site. And while MSL commands are still written in "RML", the underlying XML format changed significantly for the new mission.

Despite the differences, there was also enough commonality to allow for inheritance of some of the concepts of the tool suite, particularly the more innovative aspects cited above such as a table for sequence management and an annotated sequence tree. And as had occurred with MER, individual scripts would be combined into superscripts, particularly in areas that were more command-line accessible such as report generation for the final two phases.

Modern new primary programs presented new opportunities for CoUGAR tools. A database was created for MSL to store and manage sequences, but the only non-GUI interface available was a simple REST interface. So an intelligent command-line CoUGAR tool enabled both uploading to and downloading from the backend database, while a second tool manages the list of which sequences are currently onboard the spacecraft.

One of the most noteworthy advances that can be seen in the MSL tool suite is in "standing on the shoulders" of what's been done before. For instance the Sequence Management Table for MER is a comma-separated table of values, and each tool which accesses it needs to know the exact column of concern. For MSL, the Upload Management Table (so named because MSL radiates non-sequence files alongside the command sequences) is stored in a (far more appropriate) XML format, while retaining all the functionality of the SMT and allowing the tools to use established XML libraries for standardized data access. The rewritten time conversion tool began life with a fully-fledged host of options, rather than having functionality added in piecemeal as circumstances demanded. And a parallel tool, allowing for much faster "time math" from within other scripts, was designed without the periodic maintenance requirements that the MER version required. Reports that were invented for MER are being used with MSL, but have provided even more information and greater benefits towards accelerating the uplink process.

Perhaps the most prominent example of using hindsight to its best advantage is also one of the most powerful new tools for MSL. Originally during MER development, a program was provided to the mission which performed a "reverse translation" of the final product of the uplink process, the radiatable files (which take the binary results of the command compilation and add several additional layers of binary encoding). This program provided a complete

decompilation of every byte of the file, whether you wanted it or not. And there were many CoUGAR tools which needed *some* of that information, but not all of it, and suffered the performance loss as a result. Knowing this, a single "kitchen sink" tool was created for MSL which met every requirement that had previously been identified and more, while only doing the minimum amount of work necessary to provide the requested output. And given that this tool is homegrown, it has also allowed for newly-identified requirements to be much more easily provided as the mission continues. It now provides so much functionality that its greatest problem is managing the readability of its usage statement, and its utility is so great that it is being used not just by both the uplink and downlink sides of the mission but also by MSL's massively-integrated uplink program as well.

X. Conclusion

Mars missions require all kinds of software. The highest-profile programs are massive undertakings which provide the means by which the loftiest requirements of the mission are met. But in between the use of these primary software platforms, a great deal of work is still required to perform the complete uplink process.

A singular combination of systems engineering and agile development enabled the creation of hundreds of small programs to address the outstanding effort. As the tool set grew, new reports and methodologies accompanied the innovations encoded in the tool suite. Continued improvements to the uplink process resulted in shorter shift times, sustainable operations scheduling, and eventually a smaller staffing level, while flexibly responding (sometimes instantly) to problems and limitations as they occurred. The MER tool suite comprises almost 250 files, virtually all of which are small programs written in a variety of scripting languages. The MSL tool suite stands at roughly half that number, still sizeable in and of itself but (thanks to the many innovations and lessons-learned from its MER sibling) able to accomplish its goals in a more compact, modern, and sophisticated manner.

In facing the Goliath-sized problem of a process that was unworkable in the time allocated, the David-like CoUGAR tool suite managed to slay all obstacles and enable more than ten years of interplanetary operations.

Acknowledgments

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

References

[1]Mishkin, A. H., Limonadi, D., Laubach, S. L., and Bass, D. S., “Working the Martian Night Shift,” IEEE Robotics & Automation Magazine, June 2006, pp. 46-53.

[2]Mishkin, A. H., Laubach, S. L., “From Prime to Extended Mission: Evolution of the MER Tactical Uplink Process,” AIAA SpaceOps Conference, AIAA, Washington, DC, 2006.