REQUIREMENTS WITH TEETH: HOW TO BITE WITHOUT BEING BITTEN

Kirt A. Dankmyer

CSOC System Administrator
Wallops Flight Facility
Building E-107
Wallops Island, VA 23337
Kirt. A. Dankmyer.1@gsfc.nasa.gov

ABSTRACT

The Standard Autonomous File Server (SAFS) has operated continuously for four years now, and has yet to require an upgrade or any form of costly maintenance, while performing better than its metrics require. It has achieved this success due to several factors. First and foremost, requirements for the system were gathered and, most importantly, adhered to and not changed during the lifespan of the project; that is, the requirements had teeth. Second, future needs were anticipated, and willingness was shown to pay a higher initial cost in order to lower costs in the long-term. (This included the acknowledgement of the importance of the oft-cited, but less often performed task of producing a large amount of clear, concise documentation, and integrating this documentation with training.) Third, careful research was done to ensure the commercial and open source products used in the design of the system were not simply affordable, but were easy to maintain in the long run. A particular emphasis was placed on establishing working relationships with the vendor representatives, project customers, and mission support personnel. In this way, we ensured that we didn't bite off more than we could chew.

This paper will provide a brief overview of the SAFS, and then examine a variety of challenges the SAFS has overcome so far. It will explain how the design anticipated and met these challenges. This paper will show that by sticking to the design requirements of the system and not compromising, even problems that were not anticipated were easily overcome and the dangers of "feature creep" were avoided. That is, we gritted our teeth when customers tried to alter our design to suit what they wanted rather than what they needed, and consistently pointed customers to alternative options available on the SAFS that could meet their "requirements," without resorting to bare fangs. (For example, we will examine how the SAFS minimized the cost of increased security requirements that were levied on the system, especially since 9/11.) This is an in-the-trenches examination of a system under the pressures of constant 24x7 use; we will explain how cost and time effort were minimized, while performance was maximized with an eye toward meeting the needs of future mission, customer, and deployment scenarios.

Particular focus will be placed on the human factor, i.e. avoiding someone else's bite, in many cases through a willingness to work with others while standing one's own ground. Being an automated system, the SAFS was designed to minimize human error, which bites everyone in the end. By sticking to the design principles of the SAFS, human error was further minimized. On the other hand, the SAFS was designed with an eye toward the areas where automation does not work, maximizing the usefulness of those humans involved in its maintenance and their ability to maintain the system, through extensive and useful built-in logging and troubleshooting tools. In particular, the design recognized the importance of being transparent to those who maintain it, and the oft-overlooked importance of good relations with the people who produced the commercial software associated with the SAFS, i.e. not biting the hand that feeds you.

This work was supported by Annette M. Conger, Pati H. Peskett, Susan K. Semancik, Jeanette L. Smolinski and the CSOC contract.

To understand the advantages of the design of the Standard Autonomous File Server (SAFS), we must first understand the purpose of that design, and how a SAFS works.

The SAFS was designed to distribute satellite data to a variety of customers without interfering with the assets involved in acquiring the data, while adhering to strict data latency requirements. In other words, the SAFS had to be able to quickly send data files to a known set of customers as soon as data was fully acquired.

Second, in addition to meeting latency requirements, the SAFS had to be able to guarantee that the data always reached the customer. The data could not be lost.

There was an additional challenge in that the data had to move from a secure, closed network behind a firewall – NASA's Ground Network – to customers on the (insecure) Internet. On top of this, the ground stations on this closed network were all over the world, from Alaska to Antarctica, so they could not easily share systems. To this end, each ground station was given a SAFS system (a "station SAFS"), which flows data to a centralized SAFS system (the "central SAFS") on the open side of the network, which then flows the data to the customers. (Because of the modularity of its design, a SAFS can flow data to another SAFS just as it would another customer.) This way, only one hole in the firewall was needed, rather than one for each customer, which could have been a serious security problem, not to mention a configuration nightmare.

Breaking the acronym into its component parts, the SAFS is "Standard" in two important ways. First, the hardware and software (with some small exceptions) are absolutely the same in each system. Second, and more importantly, the SAFS has a standardized interface for receiving and sending data. From the start, the SAFS was designed to be highly flexible and modular, keeping everything generic and standardized so it could be modified later.

The "Autonomous" part of the acronym comes into play in that the SAFS is designed to operate automatically, with no operator intervention. Once a file is put on a SAFS, it is then delivered to all customers automatically.

Finally, the SAFS is a "File Server". Every SAFS system has, as an integral component, a Redundant Array of Independent Disks (RAID). The RAID is used to store the data for a certain amount of time – usually 48 hours – in case of some sort of failure. Since it is a central point of distribution, the central SAFS has a much larger RAID than the other SAFS systems.

A major part of the SAFS is a Commercial Off-The-Shelf (COTS) product called FASTCopy, which enables, among other things, the secure, reliable, and quick transfer of files.

The SAFS was originally designed, tested and documented by Susan Semancik and Annette Conger as a NASA project, and then transitioned to the Consolodated Space Operations Contract (CSOC) for sustaining engineering. Six months before the transition, I became the SAFS system administrator. After the transition, the design finally had to show its teeth, as it met the bared fangs of 24/7 "real-world" operation without aid of the developers.

Since that time, the SAFS has outperformed its requirements. The SAFS has operated continuously for four years, delivering up to 3 Gigabytes of spacecraft telemetry daily to researchers, flight operations teams, and other end users in a variety of locations worldwide. This is largely due to the advantages in the design.

One of these advantages was in the requirements themselves. All too often, requirements for a system shift and expand during development, or are not sufficiently clear before development begins. Such things tend to bite you in the long run. To prevent this, the designers of the SAFS made sure to research the requirements thoroughly, talking to all potential customers about their needs, and engaging in repeated discussions to nail down specific, clear requirements. Once requirements were set, the SAFS team absolutely did not change requirements once the design was completed.

This willingness to plan ahead extended to all aspects of the SAFS. This effort, in the long run, resulted in a significant cost savings. Every effort was made to anticipate future needs, and a willingness was shown to spend money, in terms of both equipment and hours of labor, in order to ensure a greater savings in the long term. The idea that extensive pre-planning pays off in the end is not a new one, but one which is often overlooked. The SAFS is an example of how successful such preplanning can be.

There were five major areas where planned up-front costs made a difference.

First, there was the planning of the RAID system. The designers made sure that the size of the RAID not only meet the anticipated needs of the SAFS, but greatly exceeded them, so that there was room for expansion. Also, the designers chose a vendor (Data Direct Networks) that they developed a close relationship with, one that provided excellent and personal support for the system. It is not possible to over-emphasize the importance of the human factor in this respect. Good support, in an operational environment, more than exceeds in value any savings that can be gained by going with cheap, poorly-supported hardware.

Second, there was server hardware itself. A UNIX server produced by Silicon Graphics Incorporated (from the Origin series of servers) was chosen to be the server component of each SAFS, because of the long-standing reliability of UNIX systems in general and SGI systems in particular in the server world. (Not to mention the ease with which a UNIX system is integrated into a TCP/IP network.) This hardware is capable of supporting multiple processors, allowing for future expansion and ensuring that a SAFS is capable of processing all the different streams of data entering and exiting the machine. In addition, the SGI UNIX hardware is capable of communicating very quickly with the RAID system – there is much less of a bottleneck for data access. Again, the designers had a close relationship with the vendor, which has a history of excellent and responsive support. An element of this support worth mentioning separately is the documentation – SGI provides extensive paper documentation, as well as having all its documentation available on the web. These advantages were integral in the design decision to use SGI hardware for all SAFS servers.

Third, there is the matter of the documentation of the SAFS hardware itself. Again, the importance of documentation is not a new idea, but it cannot be emphasized enough. The designers of the SAFS not only produced high-quality documentation, but they made sure it was up-to-date, up to the very last minute, right before the transition to CSOC. They produced documentation for the operators that needed to send data to the SAFS. They produced documentation for the system administrator. They documented the interface so others could integrate that information into their own documents. Every effort was made to make the documentation as clear as possible, and even accessible to those who had English as a second language. Every procedure that the SAFS team found useful in creating or maintaining the SAFS was captured. (In fact, these procedures were so useful that I still use some of the SAFS procedures when working on other, unrelated, SGI systems.)

Fourth is training. Again, all too often, a system is handed to a contractor with no guidance as to where to start. In the case of the SAFS, I worked closely with the SAFS team for six months before the transition. The SAFS team cared enough about the system – and me – to get me involved immediately, to answer my questions, to take me out in the field to install different station SAFS systems as a team. I was directly involved in the polishing of the documentation that was to go to myself and others. In terms of planning, the SAFS team had prepared themselves for this period of training long before I appeared on the scene, and were willing to put in the labor required to get me up to speed. The spirit of contractor/civil servant cooperation that was supposed to be the hallmark of CSOC was strong in the SAFS team.

Finally, we come to spares. This is one area where an initial outlay of money can save an immeasurable amount later. Every SAFS has a backup clone of the server, not to mention spare RAID drives and spare parts for every single component of the SAFS. Every ground station – and the Goddard Space Flight Center, where the central SAFS is located – has these spares on-hand and ready to use.

All of these examples of pre-planned up-front costs will be of relevance later.

Another reason for the success of the SAFS system was the planning that took place with regard to the COTS products and the planned use of open-source products. Where no known open source or standard solution existed, the best possible COTS product available at the time was chosen on its technical merit. Wherever possible, however, tried-and-true methods were used. A large amount of the SAFS was written in shell scripts, which have been around for nearly thirty years, and have seen extensive use throughout the UNIX world. Long before security was the issue it is considered to be today, the SAFS team recognized the need for security as well as remote administration, and installed a tested, open-source version of Secure Shell (SSH) on all SAFS servers for use in remote administration.

One final reason for the success of the SAFS involves the careful planning of the transition from NASA to CSOC itself. In addition to the training mentioned before, every effort was made, before the final transition, to make sure every possible objection or concern from every side of the fence was considered and responded to, and that all stakeholders, from customer to designer, had a say. Linking into the importance of documentation and requirements, it was made crystal clear during the transition what was expected out of CSOC with regard to the SAFS.

I could go on about the design until we are all very long of the tooth, but the advantages of the design are best illustrated in terms of examples – in particular, in terms of several of the challenges the SAFS had to face.

By its very nature, the SAFS is designed not to support a single project, but to provide data to a variety of projects. With each project, new requirements were levied upon the SAFS – which the design anticipated and dealt with.

The first project to use the SAFS – indeed, the project that SAFS was tested with – was the Quick Scatterometer project (QuikSCAT). A variety of customers, from Jet Propulsion Labs to the National Oceanic and Atmospheric Administration (NOAA), were interested in the data coming out of the QuikSCAT project. Therefore, as mentioned before, from the start the SAFS was designed to be capable of sending data automatically to multiple customers on different platforms (Microsoft Windows as well as UNIX), using built-in features of the FASTCopy COTS product. Also, in another example of good design, the configuration changes required to add a new customer are relatively small.

QuikSCAT set the tone for stringent data latency and reliability requirements, a challenge which the design anticipated and met. Again making use of the native abilities of the FASTCopy product, the SAFS attempts to send a file three times to a given machine, and, if that fails, it then fails over to a secondary machine, specified by the customer, which it tries to send data to three times as well. Failing that, an email notice is sent to the customer that the push has failed on both the primary and secondary customer machine, explaining where on the SAFS the data can be found, so the customer can connect to the SAFS and acquire the data that way.

In other words, every effort is made to make sure the customer gets the data. Even if a file transfer is interrupted, a SAFS, again using an ability of the FASTCopy product, can re-start the transfer where it left off when it tries to re-send – no data is lost in the process.

Because of this, the SAFS has nearly always met its data latency requirements for QuikSCAT, and has continued to do so even after other projects started to use the SAFS. The only times data latency requirements have not been met is due to issues unrelated to the SAFS (operator error, network problems, et cetera) or due to the swap space problem, a challenge I will discuss later.

Here, as with future projects, the flexibility of the SAFS was key. The SAFS was designed not just to meet the needs of a single project, but with an eye toward a variety of later needs, even ones that no project was currently demanding. For example, because the SAFS is designed to be capable of receiving data from anywhere and sending it anywhere, when the QuikSCAT project wanted to be able to send a processed data file to the SAFS to be distributed to the Japanese space agency, NASDA, it only took a small configuration change and some testing to get this mechanism in place. Given this flexibility, each ground station needs only one SAFS to do its job, even with multiple projects, files, and customers involved.

Several more examples link to the Advanced Earth Observing Satellite II (ADEOS II) project, which was highly instrumental in the creation of the SAFS in the first place. While the QuikSCAT project was the main project that was used to test the SAFS at first, the designers knew that the ADEOS II project was coming down the pike, and that it wanted to use the SAFS. They also knew, through their research, that NASDA, a major ADEOS II partner, had some very specific requirements in terms of how the data was to be sent to them, and what protocols were to be used in notifying them of that data. In particular, a Data Ready Notice (DRN), in a very particular format, had to be sent to NASDA to notify them their data was ready. After the data is received, a Data Receipt Notice (DRN), again in a very particular automated format, is sent back.

Two all-too-common reactions to stringent requirements are to gripe about them and/or attempt to work around said requirements. Instead, the designers of the SAFS embraced them. Not only did they design the SAFS to use the data notification protocol that NASDA required, but used a version of the protocol to perform notifications for other customers, and coupled the DRN/RCN system closely with a logging system, so it was easy to tell at a glance who received what file, and when. Because of the work done in finding out the needs of the customers, the SAFS design not only met the needs of the ADEOS II project, but provided a useful logging and data notification facility that is now used by all projects that acquire data from the SAFS. This ability, through the logs, to know whether files had arrived and where, made it much easier for all projects to troubleshoot and to monitor the flow of their data. Even before ADEOS II was launched, this ability was used extensively in testing and by the QuikSCAT project.

Also, as the ADEOS II project was gearing to launch, the importance of having requirements with teeth came to the fore. As an administrative tool, the SAFS team had written a program that processes the logs produced by the notification system into an easy-to-read web page. Several times, members of the ADEOS II project or the QuikSCAT project tried to turn the regular production of the web page report into a *de facto* requirement.

At this point, however, the SAFS had already been transitioned to CSOC – development was over, and the requirements had already been set. A lot of pressure was brought to bear regarding this web page. However, while remaining polite, the SAFS team (consisting, at first, of the developers and myself, and later consisting of just myself) remained firm on the fact that the production of a web page from the SAFS logs was not a requirement of the SAFS system, and would not be supported as if it were mission-critical data.

Often times, when something like this comes up, the sustaining engineering group bends to the wishes of the project, resulting in untold clandestine hours of after-the-fact development – and cost overruns. In order to get the reliability for the web page that was desired by the customers, a dedicated – and redundant – web server would have had to been put up, with appropriate personnel to maintain it. In addition, the ADEOS II team wanted an additional SAFS web report which did not at that time exist (and still does not exist), which would calculate the latency data for all the files, rather than simply showing whether the files had arrived or not, as the current report did. This would have required many man-hours of additional development, on top of the costs already mentioned.

Also, there were technical issues to consider. The SAFS was not designed to produce web reports – that feature was an afterthought. It was designed to move files to customers. Any time spent doing something other than sending files to the customers must, by necessity, come out of the time and processing power normally dedicated to the primary function of the SAFS.

When it became apparent that CSOC could not pay for such an after-the-fact effort that was not supported by the system requirements, and when it was made known to the ADEOS II and QuikSCAT team how much such an effort would cost, the issues regarding the web reports largely disappeared, almost overnight. This situation came about after extensive good-faith efforts on the issue and much polite correspondence, while remaining firm on the requirements of the SAFS. No fangs were bared.

The lesson here is that while it is important to please the customer, it is human nature to try to get more out of a system than it is designed to do – and more out of support personnel than they are paid to do. However, so long as the customers are treated with respect, it is possible to disagree with them, and still get work done. It is important to note that while this issue was being discussed, the primary purpose of the SAFS continued unabated – files continued to reach the customers. The customers, honestly, had nothing to complain about.

Plus, the dangers of feature creep are well known; there is always a distinct possibility, in trying to produce a product that does everything, that the result is a product that does nothing. I used to joke that just because the SAFS does not make coffee, and a customer would find it convenient for the SAFS to make coffee, does not mean that one must build a coffeemaker into the SAFS. It should be possible to say "no."

On a more trivial note, another capability the SAFS has that is used by ADEOS II involves file naming. The SAFS has a standardized file-naming convention that it uses to help keep track of important information about the file, like when the acquisition of the signal from the satellite occurred, down to the second. However, knowing that ADEOS II and other, forthcoming projects would want to use their own file-naming conventions, the SAFS was designed to be capable of handling this, while still providing (internally) the information that the SAFS needs.

Another small example of the advantages of the SAFS design connects to the file naming convention and goes back to the overall flexibility of the SAFS. The preferred method for acquisition of data on the SAFS is the FASTCopy push, as it is automatic and reliable, i.e. using FASTCopy to send the data directly to the customer. However, again going back to knowing one's customers, NASDA preferred a situation where they pulled the files via the standard UNIX File Transfer Protocol (FTP), using the Data Ready Notice to know when to connect to the server. Therefore, this second facility was added to the SAFS, as it would not damage the way the SAFS operated, and it added some additional flexibility to the SAFS. This was a different way the SAFS could get data to the customers, or, more accurately, that the customers could use to get their data themselves.

(It is important to note – and this will become relevant later – that an FTP push was not implemented because it would have required writing from scratch, for FTP, all of the tools for data reliability that already existed in FASTCopy, resulting in a lot of extra development for very little gain, and possibly not even the same level of reliability.)

When the High Energy Solar Spectroscopic Imager (HESSI) project came along, it did not have the money to buy a FASTCopy client. Therefore, the FTP method was perfect for them – they would get a notice when the file was ready, and then FTP it. In addition, the HESSI project had no particular naming convention it needed to use, so the fact that the SAFS already had a standard naming convention saved them a lot of time and effort. (Again, this is a small example of the advantages of being willing to do the additional work up-front, during the design process, to produce cost savings later on down the road.)

Finally, for a very long time there has been talk of having some older missions – satellites which were launched before the SAFS came along – distribute their data to the customers via the SAFS system. One obstacle to this is that most of the missions in question received their data on the closed side of the network, so it would be impractical for the data to be acquired from the central SAFS.

Again, the flexibility in the design of the SAFS came to the rescue. As far as a SAFS is concerned, any customer is the same as a SAFS. The station SAFS, with a trivial configuration change, could be set to send data to a customer on the closed side of the network just as easily as it could send data through the firewall to the central SAFS. (We will talk more about these legacy missions later.)

Overall, as each new project was added to the SAFS, they brought with them new requirements – but nothing that had not already been anticipated. So long as the potential customer understood what the SAFS was for, i.e. that it was for delivering files to customers, not for making coffee, then the SAFS was a suitable tool for the task. These examples only touch on the different projects that have made use of or considered making use of the SAFS.

Another battle that the SAFS had to face involved the station operators who fed data to the SAFS. Traditionally, operations personnel had full – in many cases, administrative – access to the inner workings of a given system. This was because many systems require troubleshooting in the field.

However, the SAFS was designed from the start to be autonomous, so by definition it had to be as "bulletproof" as possible. It should be capable of operating with no intervention whatsoever.

It is another unfortunate fact of human nature that given the opportunity to tinker with something, someone usually will. The chance of this happening increases exponentially the more people who have access to the machine. I know from bitter experience with certain non-SAFS systems that I maintain that this is true. If I had a dollar for every time a problem was the result of someone changing a configuration file and not telling anyone about it, I wouldn't have needed CSOC to fund my trip to this symposium.

Also, there is a security issue. Operations is a 24/7 game, with constant shift changes, not to mention changes in personnel as part of the normal business hiring cycle. Long before the policy of limited access was regularly enforced at NASA, the designers of the SAFS realized that having, for example, a single "operations" account that several different people used was a security nightmare, and having a separate account for every operator was a nightmare of a different nature, considering the sheer number of personnel involved. Therefore, from the very start, the SAFS was designed to be directly accessed by only one person – the system administrator, who would only do so in order to perform system maintenance.

This did not mean that station personnel did not want access. In some cases, they demanded it. Once again, however, a polite but firm attitude and good design saved the day, without the need to resort to tooth and claw. Going back to the requirements, the designers of the SAFS made sure that it was understood that operator access endangered the autonomy and security requirements of the SAFS.

That done, the design anticipated the needs of operations, and provided for them. Did operations personnel need the SAFS logs? They could access them via FTP. Did station personnel need to know if the data got through? That was the purpose of the SAFS web reports mentioned earlier, and, if those did not work, one could get the same information by examining the logs via FTP. Did administrative work need to be done in a remote location, such as Alaska? The SSH program provided a secure (encrypted) method of remote administration that could be used by a single system administrator at the Wallops Flight Facility.

And while we are on the subject, let us touch on the matter of security. As all of you know, computer security has been of increasing concern over the years, and has only become even more of a priority since the 9/11 tragedy. The design of the SAFS anticipated this.

First, there are the security advantages already mentioned – limiting the number of users accessing the system to the system administrator only, and using SSH to encrypt all remote administration traffic. On top of this, FASTCopy has several security features. First, it has the capability to create a password for its use alone. This password, which is only sent in an encrypted format, is only good for sending and receiving files. Even if the encryption was somehow broken, one cannot access the SAFS system proper using the FASTCopy password. Second, it has a built-in access control facility; by default, no one can push or pull files using FASTCopy. Every customer is added explicitly to the security access list – not only does the customer have to have the right password, but must be connected from the correct place. All of these parameters are configurable – one can set up FASTCopy to only allow a pull from a particular machine with the correct password, and given only certain rights on the system.

Also, the fact that the SAFS was designed atop a UNIX system using as much standard and open-source code as possible is important. UNIX has been around for over thirty years. Methods and tools for securing a UNIX system are well-known and mature. Second, there is an active, thriving, and large community working on UNIX security issues, and an administrator is not dependant on one entity – such as Microsoft – to produce fixes. Finally, because the SAFS relies on such tried-and-tested methods as shell scripting to do its work, security does not "get in the way" of the operations to the SAFS – security patches can be applied, services can be turned off, and the operating system can be upgraded, without affecting the normal operation of the system. I know this because as security requirements have tightened, the SAFS has continued to weather these changes and more with no problems at all.

As a counterexample, I maintain a few machines which are, in terms of server hardware, exactly the same as the SAFS. When I applied a routine security patch to one of these non-SAFS systems, a patch which required upgrading the operating system and had been tested on the SAFS, the non-SAFS system broke – the software on it stopped working, though it booted up fine. I found out later that the system in question used proprietary hardware and proprietary, Java-based code that was not compatible with the latest version of the operating system. The many, many man-hours I spent dealing with this issue contrast starkly with the ease with which I upgraded and patched the nearly-identical SAFS. Sometimes the only way to know good design is to notice the problems that the design is *not* causing.

Leaving security behind for now, one major obstacle the SAFS had to overcome involved a challenge which came to be called the "swap space problem". Without going into too much detail, an unfortunate interaction between the FASTCopy setup on the SAFS and on a customer machine resulted in a situation where if a file with the same file name is sent more than once, problems with the swap space on the SAFS can result which can, in time, crash the SAFS, and will certainly, before then, block data flow.

However, it took months of man-hours and extensive work to find this out, as the cause of the problem was not obvious – the problem was insidious, and it often took days after the triggering event to develop any symptoms. Here is where the initial design of the SAFS played an important role. Aside from the technical issues that prompted the use of the FASTCopy product, as alluded to before, the other reason the SAFS team chose FASTCopy is because that COTS product is produced by a small, competent team of technologists who were eager to do business with NASA. Because the size and nature of the company, the SAFS team was able to develop a personal relationship with the vendor which became critical in solving the so-called "swap space problem".

Again, the importance of this human factor cannot be underestimated. The dogged, tireless support we got from the FASTCopy people is nearly (but not quite) impossible to get from an entity as large as, say, Microsoft. This fact was factored into the design – in fact, every element of the design was informed by the concept of making it not just easy to put together, but easy to maintain. And the heart and soul of sustaining engineering are the people who do it, and the relationship with the vendors involved.

However, this said, one must remember that good customer relations and good design can only go so far. Another, unfortunate lesson one needs to learn is when to know something is impractical.

Earlier I mentioned certain legacy missions that were considering acquiring data from the SAFS. These missions were used to getting their data from a buggy, difficult-to-use system which pushes data via FTP. The main impetus to move these legacy missions to the SAFS came from the ground station operations group, upon whom the burden of getting this system to work, often with the software equivalent of spit and baling wire, fell.

Many man-hours were spent on this concept, of moving from the less-reliable system to the more-reliable SAFS. However, there was one major stumbling block. The missions did not want to buy FASTCopy, and did not want to pull their data via FTP – they wanted it pushed by FTP, the one thing the SAFS will not do, for reasons already discussed. (That is, the additional development would have been more expensive than FASTCopy, and would have risked breaking the SAFS.) Eventually, although discussions are still underway, it was generally decided that moving those legacy missions to the SAFS was not feasible without more money. In the end, one has to be aware of one's limitations, and work with them.

How is this cautionary tale related to design? First of all, the flexibility of the design allowed us to offer a wide variety of solutions to the potential customers, to the point where the only obstacle that mattered was the desire for an FTP push. Secondly, because the requirements had teeth, because we were unwilling, as it were, to modify the SAFS to make coffee, we saved ourself from committing to an effort that would have likely been costly and ultimately futile, given the incompatible requirements levied by the projects in question. Without the clarity of the design requirements, it would have taken a lot longer – and more money – to reach this conclusion.

In contrast, remote users willing to purchase FASTCopy are able to seamlessly integrate with SAFS. For example, commercial ground stations in Alaska and Norway were brought in to provide supplemental services to the Ground Network. They integrated FASTCopy into their existing architecture without the need to duplicate the SAFS hardware platform. Since a SAFS does not care whether it gets data from a SAFS or some other architecture, these ground stations were able to send data to the central SAFS, which was then distributed as normal to already-existing customers. Once again, the flexibility of the design was key.

These are only some of the many examples of how, in the field, that good, pre-planned design saved money in the operation of the SAFS. There are many other cost-saving elements of the design that deserve mentioning.

The SAFS team built many methods into their design to reduce basic maintenance costs. First, as mentioned before, by purchasing spares for all important components the SAFS team prevented a single point of failure, meaning that when there is a problem, there is no costly damage to operations – for example, if the central SAFS goes down, the station SAFS machines automatically send to the backup central SAFS instead, with no costly interruption of data flow service. By allowing for remote operation, the number of people needed to maintain all the SAFS systems is reduced, and this staff then reaps the benefits of specialization, in terms of being able to understand and troubleshoot the system. By logging the routine operations of the system, keeping an eye to what is required of the SAFS, the design allows the system administrator the ability to troubleshoot quickly and efficiently, and puts useful operational data right at his fingertips. The time (and, therefore, money) saved by this would be difficult to measure, as the amount would be so large. Similarly, the web-based automated reporting tools designed into the SAFS allow the administrator to access much of this information without even logging in. Plus, since the SAFS is a UNIX system, it can use the "cron" feature to automatically schedule tasks, freeing the administrator from routine maintenance, and saving further money.

Let us not forget, however, the all-important human factor. My work in the field during the transition allowed me to understand the operational environment, to meet and empathize with the operations personnel and vendors, and to understand and become sensitive to the needs of the people doing the real operations work. Anyone who has ever been involved in the sort of problems caused by a misunderstanding between vendors, operators, engineering, and customers can understand the amount of money saved, in terms of time alone, by, from the start, making an effort to prevent such misunderstandings.

The other side of this human favor lies in avoiding feature creep – the aforementioned unnecessary addition of new features to already-stable software – by having clear and technically possible requirements, with hard and fast criteria for meeting those requirements which are not dependant on pleasing a particular person or class of people. The best tools for preventing this creep, aside from simple human empathy and politeness, the value of which cannot be underestimated, include documenting one's methods and the reasons behind them. The documentation and training made it clear not only how the SAFS worked, but *why* it worked the way it did. Never underestimate the advantage of good, solid reasons behind your design decisions, and making those reasons widely known. People are much more understanding when they have access to your reasoning. Considering future needs, and providing the ability to meet them, prevents having to add a needed feature later – which links back into not just understanding the requirements of the system, but projecting those requirements into the future. And, at the risk of repeating myself, there is nothing better than a solid, close relationship with everyone – customers, vendors, operators – involved with a system. Anyone who thinks system administration is a purely technical venture is fooling themselves.

As I mentioned before, a willingness to spend additional money at the design stage saves an immense amount of money later. By picking a RAID system and server architecture that not only meets, but exceeds requirements, the

SAFS team made sure there was plenty of margin for error, and plenty of room for expansion. Similarly, a willingness to invest time, as well as money, in a good COTS vendor pays excellent dividends later, no matter how cheap the competition might be. In addition to this, I have not even begun to touch on the money saved in terms of the time required to come to an understanding of the SAFS system by the initial outlay of time and effort the SAFS team put into documentation. On average, I estimate problems have been solved literally an order of magnitude faster with the provided documentation than the time it takes me to solve similar problems on more poorly documented systems that I administer. Finally, and most importantly, by working directly with the development team, even on the tail end of the design, I was able to make sure not only that the design fit the needs of sustaining engineering, but that as a system administrator I was ready for the challenges that were to come.

After the transition, two major techniques were employed in saving costs. First, as previously discussed, one must stick to the requirements, so one is not bitten by unexpected new work. Second, my training taught me to brainstorm new solutions, to make use of the advantages of the design to come up with solutions that meet customer needs, even if those needs are not met in the way the customer originally suggested. Always replace an obstacle with an alternative solution.

In the end, not to be too toothy about it, it comes down to the old aphorism: “You need to spend money to make money.” And if time is money, well, this goes for time and effort as well. If anything, my experiences in the field has shown this to be true over and over again. We’ve known this all along, but we’re reluctant to admit it, as it means more work, and the easy way is so very tempting. Therefore it bears repeating until my teeth fall out: By spending extra time and care on the design, with an eye not only towards solving immediate problems but toward making the system easy to maintain as part of the ever-so-important human factor, the cost savings for everyone is so great that we cannot afford to *not* design things this way, lest we end up bitten on the posterior by the technical issues compounded with the eternal conundrum of human nature.