Developing Interim Systems

Jennifer Caetta – Jet Propulsion Laboratory

ABSTRACT

One of the recent challenges in the aerospace industry has been to smoothly transition operations-oriented computer systems to meet increasing demands on smaller budgets. Sometimes the best solution is not affordable, but the current situation is equally untenable. Change becomes necessary, but is only acceptable at minimal operational impact and financial costs.

Frequently, interim software solutions must be developed while new hardware is under design or production. These interim systems are characterized by an immediate need with a limited budget. This paper discusses several factors involved with producing a working software set in minimal time, such as the initial approach, budgetary concerns, and resource-scavenging. By recognizing the importance of these factors, interim system developers will be able to produce working systems with a minimum of setbacks.

Introduction

Software should be perfect. Networks should be capable of handling multiples of the maximum expected traffic levels. Hardware should never fail. But "should" will not release a system before an operational deadline, and it invariably costs more than your budget can afford. The solution is to instead create a system which meets immediate customer demands, yet leaves an open path for future modifications.

The ground data systems Jet Propulsion Laboratory range from state-of-the-art to nearly obsolete. These older systems have no funding for hardware upgrades (replacement hardware is expected within five years), yet they still support spacecraft on a daily basis. This paper focuses on methods by which the operational capabilities for these systems can be significantly extended at a minimum cost.

These methods consist of three main stages: initial approach and design, working within and around budgetary limitations, and resource-scavenging. While these methods may seem obvious, the impact of not following these steps is usually only understood in retrospect. Therefore, this paper will set forth guidelines in each of these areas and examine their usefulness with three operational interim systems (currently in use at JPL for their Radio Science Systems Group).

Program Descriptions

Getting Real-Time Data: get_tss

The first of the three interim systems which will be used in later examples is a method of getting real-time data from the central communications facility at JPL (the GCF) through a firewall and into a display package which ran on a Sun 4/330. The original system used a dedicated modem line to a Prime 4050, which filtered the data based on spacecraft and receiving station. The data was then sent via TCP/IP to a Sun 4/330 and processed into shared memory. Users could then display various parts of the data. In order to change the filter to use a different spacecraft or receiving station, the program had to be stopped and restarted with different parameters.

When the interface was to be formalized between the Radio Science Group and the GCF, the possibility of using Ethernet was considered politically unacceptable. It was decided that X.25 could be used to replace the modem interface, because it was already in use in several places on-lab. A filtering program was written for the incoming data which would allow filter parameters to be changed on-the-fly, and the routines to handle data receipt were isolated from the rest of the software – to facilitate future changes in the incoming data protocol. This system worked reasonably well; however, differences in Sun X.25 and Encore X.25 (the GCF hardware) soon demonstrated a problem: when our system was not listening to the Encore, the packets were dropped "on the floor" in the GCF, much to their dismay.[1]

It was soon suggested that management reconsider the possibility of using a UDP/IP connection, outbound data only. Upon their eventual approval, the adaptation was simple. Routes were created to allow only one-way traffic on a single unix socket, and the X.25 receiving software was modified only to the extent of creating UDP/IP sockets instead of X.25 circuits. Figure 1 illustrates the three data paths used during the transition.

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

[1] Note: this is not a problem in Sun-to-Sun X.25 connections.

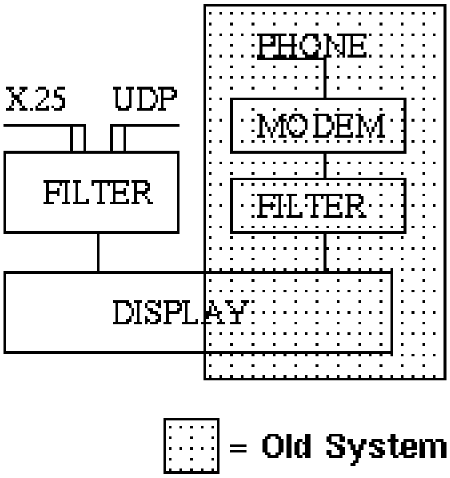

Figure 1: get_tss, original system and current system.

Remote Control of DSN Hardware: remops

The second of the three interim systems remotely controls the open-loop receiver (a digital signal processor with several physical components, commonly called the DSP) at the Deep Space Network (DSN). It allows users at JPL or Stanford to configure the DSP as well as to set up unattended operations-based configuration and run scripts. The DSP includes MODCOMP 2000 and several dedicated boards, running REAL/IX (a real-time Unix-based OS). The replacement hardware for this receiver will be fully funded for the Cassini project, but not until 2002. The unattended operations capability was initially implemented in order to support a grueling 3-year, 8,000-occultation mapping phase by Mars Global Surveyor.

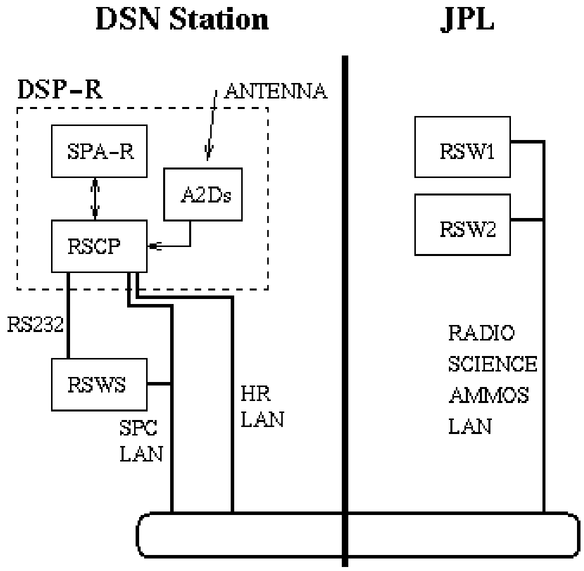

Figure 2: remops Hardware Layout

The operational "remops" system uses a Sun Sparc 5 connected via an RS-232 serial cable to the MODCOMP, using the diagnostic terminal connection as a "back door" into the system. A fairly standard client/server package handles interactions with remote users. Figures 2 and 3 show this layout.

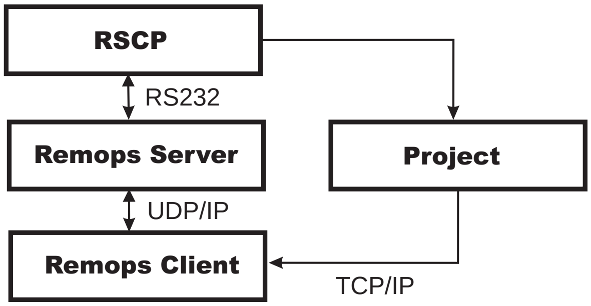

Figure 3: remops Software Data Path

Data Analysis: RSVP

The last of the three systems is a data analysis tool which originally was meant to replace the functionality of a large number of Fortran programs which ran on a Prime 4050 with an array processor. Many of these programs were simply copies of another program with minor modifications or refinements – spaghetti code with names like "resid01", "resid02", and "resid03" to distinguish the versions.

The overall structure of the Radio Science Validation and Processing tool, (rsvp), is a C++ "shadowbox."[2] Figure 4 shows the overview of the design. The front end, which must read in many different data formats extracts the required data and makes it available to any of the successive programs. The types of processing were categorized, and their arguments incorporated into the GUI (graphical user interface), but the programs are called with system[3] calls. This not only allows the scientists to continue modifying their programs, but it allows the program to remain stable in overall structure. New data formats only require another C++ class and a few header file entries; the rest of the code remains fixed. New programs require only a new system call and a GUI-incorporated argument list. Version numbers are kept on the scientist-provided subroutines, and only one version is "supported" at a time. However, multiple entry points into the program are available to accommodate the occasional need to run non-supported programs on the data, then re-insert that data back into the program.

The Initial Approach: How Does Your Project Grow?

The most crucial stage of any new project development is the initial communication of ideas and goals, and the understanding of how the project will

[2] Shadowbox: a decorative set of small attached boxes which is hung on a wall. Knick-knacks are then placed on each little shelf.

[3] Standard C library system command, SunOS 4.1.4.

grow and evolve. Several papers and books have been written on strategies to formalize and elaborate each aspect of clearly communicating requirements and limitations [1, 5]. Software process models and win-win [2] negotiation methods are continuing to expand by accommodating "real world" behaviors [4] and becoming more generalized.[4] However, some projects – interim systems in particular – are very rarely ever "done." These systems are created to fill a basic need, and are usually expected to be replaced or even retired. But what if an interim system has years before that happens? The possibility of added capabilities and refinement at a minimal cost can be very tempting.

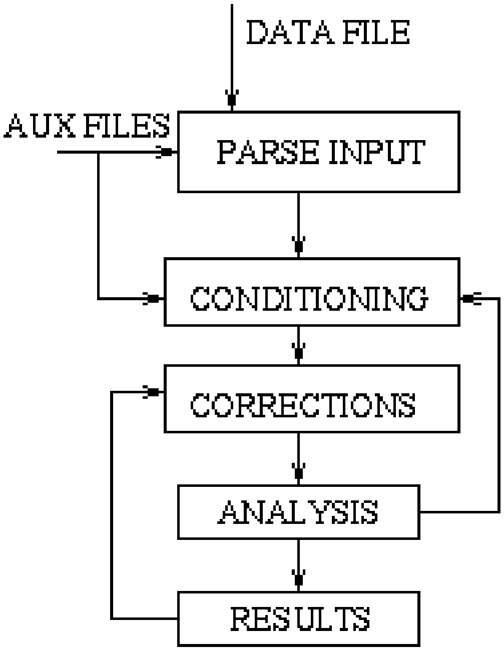

Figure 4: rsvp Software Data Path

Interim systems therefore tend to evolve into a series of stacked spiral process models [3], something which could be described as a "tornado model." Each layer inherits a base set of risks and dependencies from the lower layers, but is uniquely linked to a specific capability which is to be implemented. Capability requestors can be users, management, or "evolutionaries" – developers who wish to modify code now to make future instantiations easier to develop. The capabilities themselves usually do not interfere with the functionality of previously-developed instantiations, although such conflicts are usually identifiable through the inherited dependencies.

Because of this penchant for growth within the low-budget, fast-turnaround and high-risk[5] framework which characterizes interim systems, several of the normal developmental stages – communication, realization of risk, and designing for reusability – become high-risk factors themselves. Without an awareness of the evolutionary factors within each stage, a design could result in a program too inflexible to accommodate future changes which might have been anticipated.

Communication

The clear communication of objectives between developers and requestors is perhaps the most significant method of preventing a failure to successfully implement a software package. When developing a system which is expected to encounter new-capability requests, the communications and negotiations methods mentioned above become even more critical. By "getting close to the customers" [7], and actively searching for possible future requests, the design of a system can anticipate the need for certain types of flexibility with a much larger lead time than would otherwise be available. Weekly meetings to discuss status, problems, and possibilities can not only achieve Gilb's "risk sharing principle" [6], but makes sure that no misunderstandings have occurred and that both developers and requestors understand the overall direction of progress. In addition, the future use of the system can be addressed; the design can then inherently facilitate the eventual implementation of new capabilities.

As an example, the rsvp program, described above, was not simply a software port to a different architecture; it was a complete re-design in structure and responsibility. The package needed to smoothly accept multiple input data formats, ensure easy the implementation of new input format handling, and allow the scientists who originally developed the analysis programs to continue to develop them to accommodate new frequency band equations, new antenna combination methods, or new analytical ideas.

The "shadowbox" design of rsvp, described above, satisfied the above conditions. The design also made it easier to incorporate the most important portions of the Prime 4050 original software, thereby creating a basic-use capability quickly. However, one important design stage did remain unrecognized until well after the main structure had been completed: the value of continued discussions with the users about the difficulties they were having or what kind of functionality they would like in the future. As it turned out, some of what they were asking for was fairly simple to accommodate, yet they had written separate programs to bypass that need.

By modeling the package with the consideration of how the software would grow and develop (using a "tornado"-style model, Figure 5), this extra effort

[4] However, actually applying these models to a proposed system always requires some tailoring – even if that tailoring consists of recognizing each phase but not acting upon it.

[5] In this case, the risk associated with the uncertainty involving legacy systems.

could have been avoided. However, after adapting the software upgrade procedure to reflect this type of model (incorporating weekly meetings, feedback sessions, and future-use discussions), we found that developers were proactively tailoring the code to facilitate the eventual implementation of these "future-use" capabilities in parallel with implementing changes requested through the feedback sessions. This synchronous development did not greatly increase the time needed to complete the feedback requests, and when the decisions were made to implement some of these capabilities many months later, the time involved was much less than anticipated because of this software "priming."

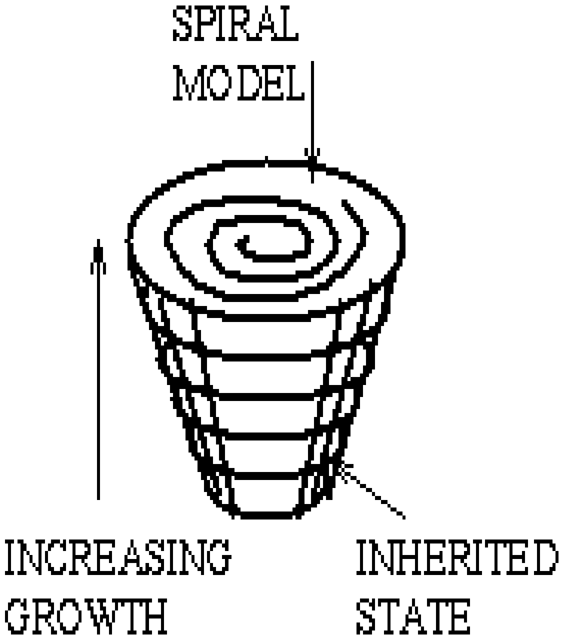

Figure 5: "Tornado" Model

Realization of Risk: Just Don't Panic

Interim systems have an inherent high-risk factor: the high probability that the system will need to interface or interoperate with a legacy system. These systems have a penchant for undocumented behavioral anomalies, or "features." Even the best risk assessments underestimate this potential.

For example, while testing the terminal interface between the DSP and the remops program, two such "features" materialized. The first was the assumption in the terminal session software that no human would sit down and type 1024 commands into the system (the terminal session was not the primary command interface, but a diagnostic back-door). However, the human would (obviously) want a printout of all the commands entered. Therefor, all commands were stored in a buffer – a buffer with 1024 spaces. Upon reaching the 1024th command, the system refused to take further input until the terminal session was aborted.

This limit led to the idea to open up a new terminal session whenever a new configuration set needed to be input (which uses approximately 15 commands). Within five minutes of back-to-back test runs, we discovered a memory leak in the DSP system queues; the more terminal sessions started and stopped, the less memory was available for the rest of the system. Eventually the entire system would hang. Therefore the only compromise was to keep track of how many commands got sent, and only when that number neared 1024, to automatically shut down and restart the terminal connection. This would give the system the maximum possible length of time in which to operate before the system would need to be rebooted: about five weeks.

Luckily, the hardware rarely was able to operate for three weeks without needing to be rebooted for other, less understood, reasons.

While it is very difficult to accurately model this type of high-risk behavior, interim systems designers do have some leverage when proposing solutions which would otherwise not be considered. Support from management must be strongly committed, especially when the proposed solutions are best described as "creative." Don't panic: interim systems projects are rarely undertaken when they are not urgently needed. While not exactly intended in the same context, Gilb's uncertainty motivation principle [6] can be applied to the potential for a lack of support from management:

> Uncertainty in a technical project is half technical and half motivational, but with good enough motivation, uncertainty will not be allowed to lead to problems.

Frequently, the thought of an interim system not working provides plenty of motivation.

Designing for Reusability

In general, the expectation upon an interim system is that it will be in use only until the replacement system arrives. However, this expectation cannot defend throwaway code, nor code which is not portable or upgradeable. The replacement system may not be available until much later than expected.

In anticipation of the change, however, designers should understand the expected replacement system, not necessarily as a final destination in the growth of the interim system, but more as a system which is trying to surpass the possibilities of the interim system. Time should not have to be spent redeveloping the modules which create old capabilities! Instead, by writing code which is easily reused and portable, modules of the interim system can be incorporated into the replacement system, allowing more time to be spent on those previously unreachable capabilities.

In this way, the interim system becomes a resource which can be scavenged later, with a minimum amount of portability and modification effort. The get_tss program, described above, exemplifies this through the X.25 to UDP/IP transition; the body of the code was unchanged while only the socket initiation code needed to be changed.

Budget Limitation: Creativity While Being Broke

The second guideline towards successful interim system development is creative financing. Usually, the most insidious of budget allocation patterns is the tendency to think of financial sources in set, well-used ways. At JPL, spacecraft projects tend to think in terms of "project money": what that money buys, the project owns completely. In companies with multiple projects, this can lead to a large amount of replication of resources with minimal gains in reliability.

At JPL, the role of Radio Science is (given a spacecraft, a signal, and a receiving station) to produce planetary/atmospheric science data. As projects come and go, the Radio Science Group needs to continually expand and grow in order to accommodate these changing needs and be a truly multi-mission service provider. Because of this growth pattern, the Radioscience Operations and Data Analysis Network (RODAN) can can itself be characterized as an interim system. However, mainstream, project-line funding can be severely detrimental to the overall quality (robustness, reliability, capabilities, and veracity) of the Radio Science system.

As an example, until last year the low number of concurrent projects had allowed RODAN to follow the same trend as the projects: each project owned a subnet off of the central operations backbone, and all workstations associated with that project were installed on that subnet (See Figure 6). Because Radio Science was not located in the same building as the project offices, however, fiber lines had been installed between buildings: one for Galileo, and one for Mars Global Surveyor (MGS). Each project was willing to buy Radio Science a single workstation for real-time operations and sequencing activities. This was all in accordance with the budget limitations of each project, and all very separate from each other.

When the operations and sequencing scenarios for Cassini were initially considered, this trend showed to be extremely restrictive. The expected future increase in concurrent mission numbers (e.g., Pluto Express, Mars 98) and the possibility of a single-point failure associated with the one workstation/one project funding scheme emphasized further the need for a major design change. Politically, however, no funding was available: Cassini was thinking of duplicating the entire Radio Science team within the Cassini hierarchy, MGS had already spent all it could on Radio Science by funding the remops development, and Galileo was settled into the status quo.

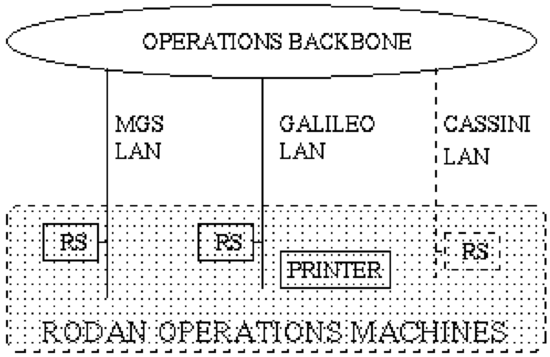

Figure 6: Original Radio Science Operations Network Design

The basic characteristics of interim systems, however, provide a unique source of justification of financial expense. Invariably, interim systems are strongly needed – and not always only by the proposing group! By combining a financial analogy of Gilb's uncertainty motivation principle with a proof of lowered future costs, along with a "sale-by-example" approach, both the Cassini and Galileo projects realized that they, too, needed the remops system. However, both projects were also made very aware of the vulnerability of the current single-point failure design, so when the final layout of a single Radio Science LAN (Figure 7), in parallel with the individual project LANs, they were more willing to trade a project-owned system for an affordable and robust shared system. MGS would give their fiber line back to JPL, Galileo would donate their fiber and hub for general Radio Science use, and Cassini would fund labor costs. The benefit for modifying the strict project-line funding would come in operations cost: with the remops system, 30 minutes of pre-pass calibration are no longer needed. At the 70-m antennas, time costs about $1200/hour, and at the 34-m stations, around $750/hr. Obviously, the savings build quickly.

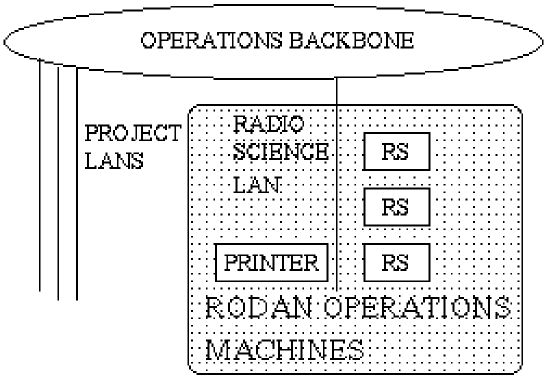

Figure 7: New Radio Science Operations Network Design

Admittedly, the feat of pulling financing out of apparently thin air does not always work out as cleanly as that example. While the diplomacy, approach, and presentation of a different type of financing remains an art form, interim systems, however, give developers a lot of leverage. For example, when dealing with the burgeoning Cassini "fiefdom" design, the Radio Science group can really only emphasize its "proven and stable" contracting firm image, and hope for the best. However, if Cassini wants the remops system to be ported to HP/UX, or utilize the operational experience which resides within the Radio Science group, then financial support would obviously be expected.

Resource Scavenging: Scotty[6] had the Right Idea

The third guideline crucial to the development of interim systems is resource scavenging – essentially a protection against wasted time. With a little time invested in identifying available resources, a lot of time can potentially be conserved. Most of this scavenging phase is already a part of the spiral software model – such as finding appropriate third-party applications. However, interim/legacy system interfaces have a high cost risk associated with not emphasizing certain types of resources – freeware/COTS applications, original developers, and creative perspectives.

Tools

The primary source of available tools is from other programmers or developers – and for interim systems, the fact that stable freeware is much more affordable than many COTS packages is not lost in the design phase. While these packages may not exactly fit the system's needs, adaptation or extension can frequently be added at a fraction of the cost of writing a comparable system.

For example, while writing a graph tool for the analysis package rsvp one of the developers was able to get a lot of ideas and examples from existing software, such as the Itcl source[7] and documentation, and the associated extended widget libraries. This saved him a lot of time; he was able to produce a working product within a few weeks.

People

Original developers of the legacy systems with which the interim system must interface are also extremely valuable. While many anomalies may not be documented, these developers may be able to give information about the problems and advice on how to work around them. Unfortunately, these developers are frequently gone or unavailable for extended periods of time, or in one case "wouldn't touch that system with a 10 foot pole, except to hit it very, very hard."

A lack of these human resources can severely impact the effectiveness of an interim system because of the close ties between legacy system interfaces and undocumented behavior. For example, while developing the remops software, the interface to the DSP proved to be troublesome because of some assumptions about how the communication between the MODCOMP and the secondary system (the Signal Processing Assembly – Radio Science, or the SPA-R) was conducted in the case of a reboot of the SPA-R but not of the MODCOMP. It had been understood that the MODCOMP would re-send the configuration settings to the SPA-R, but this was not exactly the case; the MODCOMP only sent over a subset of these configurations. This was not discovered until the system was in operational use: such a reboot pattern does not happen frequently, and it had not been tested because it technically wasn't a part of the remops system.

In these cases the next best resource is a user-group maintained log of "unexpected" observed behaviors. The documentation of interim systems frequently must cover problems with the legacy systems; with such a log, the same problem won't bite you (or another developer) twice.

Creative Perspectives: Does This Really Have To Be This Way?

The last segment of resource scavenging is to identify which applicable technical restrictions are easier to change than avoid, and which are easier to avoid than to change. Interim systems are frequently subjected to technical restrictions which are based on older systems, and which may not be not completely relevant. Therefore, the design of the system must take into account the probability of a change in these restrictions, and factor in the benefits of using the resources which would then become available.

While originally implementing the get_tss program, Cisco routers were not trusted to provide a firewall when outbound UDP/IP data was allowed; therefore the X.25 interface was selected. The design of the package, however, ensured that if an ethernet connection was eventually allowed, minimal changes would have to be made. Over the course of a year while the communications facility's staff became more familiar with the Cisco routers, and given the problems encountered with the Sun/Encore X.25 interface, the system was able to be switched to UDP/IP, and the extra cables, cards, and switches were able to be discarded. In this case, the restriction could not be avoided, and it was therefore easier to simply change the software when that restriction was eventually lifted.

On the other hand, the remops system is not able to get monitor data via broadcast at the remote workstations at Stanford due to valid security restrictions. In this case, the restriction is avoided by designing several parallel methods of receiving monitor data, and using the best available method for each machine:

[6] The engineer from Star Trek (the original series).
[7] Itcl2.0 for these packages.

via broadcast, database query, or by a special query to the DSP itself.

Conclusions

While the spiral software model has incorporated the vast majority of system development and design concerns, interim systems require extra emphasis on the initial approach, budget considerations, and resource scavenging. Because of the nature of such systems – interfacing with legacy software and fragile hardware, intended for short-term use and therefore needed on an per-capability basis – ignoring the importance of these factors can lead to serious consequences, such as late-stage design changes, unexpected incompatibilities, or worse, not having a working product if the old system fails before the new system has been funded. Essentially, interim system development theory can be summed up by the simple phrase: get it done, make it work, and don't spend any extra money. By following the above model, developers can build working, growing interim systems with just that characteristic.

Author Information

Jennifer Caetta graduated from the University of California at Berkeley in 1993 with a Bachelor's Degree in Electrical Engineering/Computer Science. She joined the Jet Propulsion Laboratory, where she is an applications developer, systems administrator, and configuration manager. Reach her via US mail at Jet Propulsion Laboratory, M/S 264-325, 4800 Oak Grove Dr., Pasadena, CA 91109. Reach her electronically at caetta@rodan.jpl.nasa.gov .

Bibliography

[1] B. Boehm, "A Spiral Model of Software Development and Enhancement", *Computer*, May, 1988, p. 61-72.

[2] B. Boehm, "Theory-W Software Project Management: Principles and Examples", *IEEE Transactions on Software Engineering*, 1989.

[3] B. Boehm and F. C. Belz, "Applying Process Programming to the Spiral Model", *Proceedings of the 4th International Software Process Workshop*, May, 1988.

[4] B. Boehm and P. Bose, *A Collaborative Spiral Software Process Model Based on Theory W*, Aug., 1994.

[5] R. Fisher and W. Ury, *Getting To Yes*, Houghton Mifflin Company, 1981.

[6] T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley Publishing Company, 1988.

[7] T. J. Peters and R. H. Waterman, *In Search of Excellence*, Harper and Row, 1982.