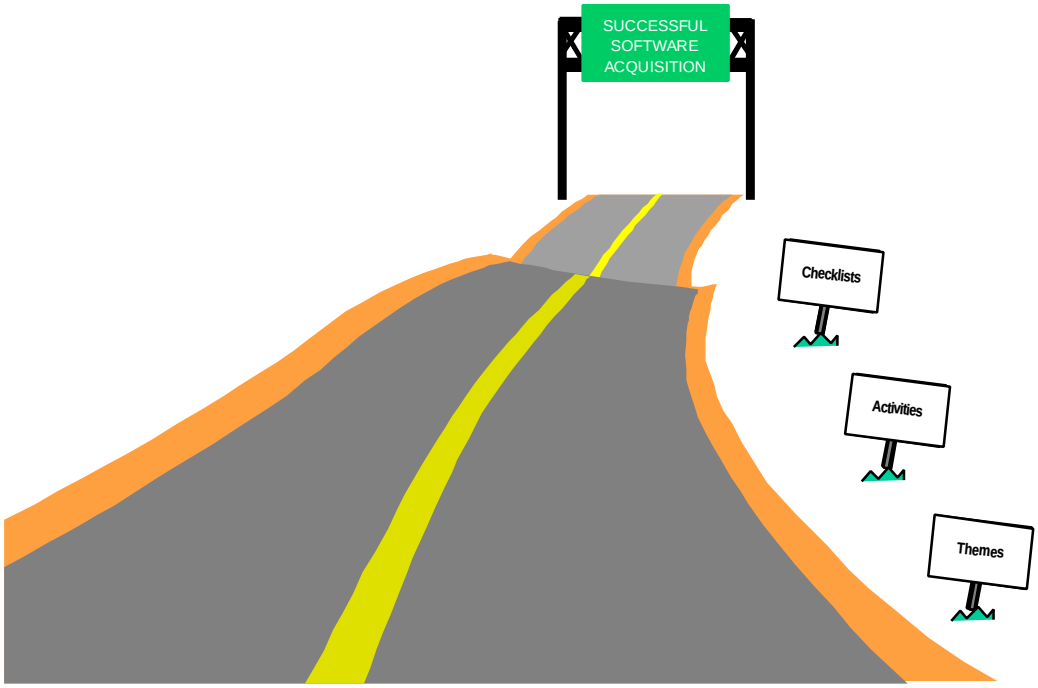
U.S. Department of Transportation
Federal Highway Administration

The Road to Successful ITS Software Acquisition

Executive Summary



The Road to Successful ITS Software Acquisition

Executive Summary

July 1998

Prepared for the Federal Highway Administration
by Mitretek Systems

EXECUTIVE SUMMARY

Software is Different

"Unfortunately, software development does not progress in accordance with the rather simple rules that govern most functions." — [Putnam and Myers, 1996]

"The odds of a large [software] project finishing on time are close to zero." — [McConnell,1996]

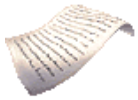
Acquisitions that involve a significant amount of software development are notorious for their problems. Missed schedules and cost overruns plague the acquisition process. When systems are finally delivered, they are often unreliable and do not meet all their requirements. Some projects are even canceled before any products are delivered.

Experienced project managers find that proven managerial techniques, which previously worked so well for them on other types of projects, fail for software. They complain about their lack of insight into what the final system will be like and their lack of visibility into progress by the contractor. "It's not like seeing asphalt being laid down." More than one manager has concluded that "software is different", that it often defies intuition gained elsewhere.

Unfortunately, ITS software acquisitions are no exception to this software norm. One senior ITS manager lamented he'd never been involved on a software acquisition that he was satisfied with.

Representatives from the public and private sectors who have been involved on ITS software acquisitions have very different perceptions as to what goes wrong. Each feels that the other takes advantage of the situation. They perceive that the other party "wins" while they "lose". In fact both parties lose: While public-sector customers face the problems cited above, private-sector contractors often lose significant amounts of money on software. This leads to mistrust. Both sides then resort to acquisition practices that further exacerbate the situation.

The good news is that there are proven techniques for managing software acquisitions. This document presents best practices, not rigid guidelines, to assist you. Use them selectively, choosing those most appropriate for your agency and project.



The top five percent of software organizations have no canceled projects, consistently control costs within 5 percent of budget, and meet schedules within three percent. [Jones, 1997]

Acquisition Themes

Figure ES-1 summarizes the themes upon which successful software acquisitions are built. Collectively they represent a different way of doing business, the response to "software is different". The themes recur again and again throughout a software acquisition and guide the various acquisition activities.

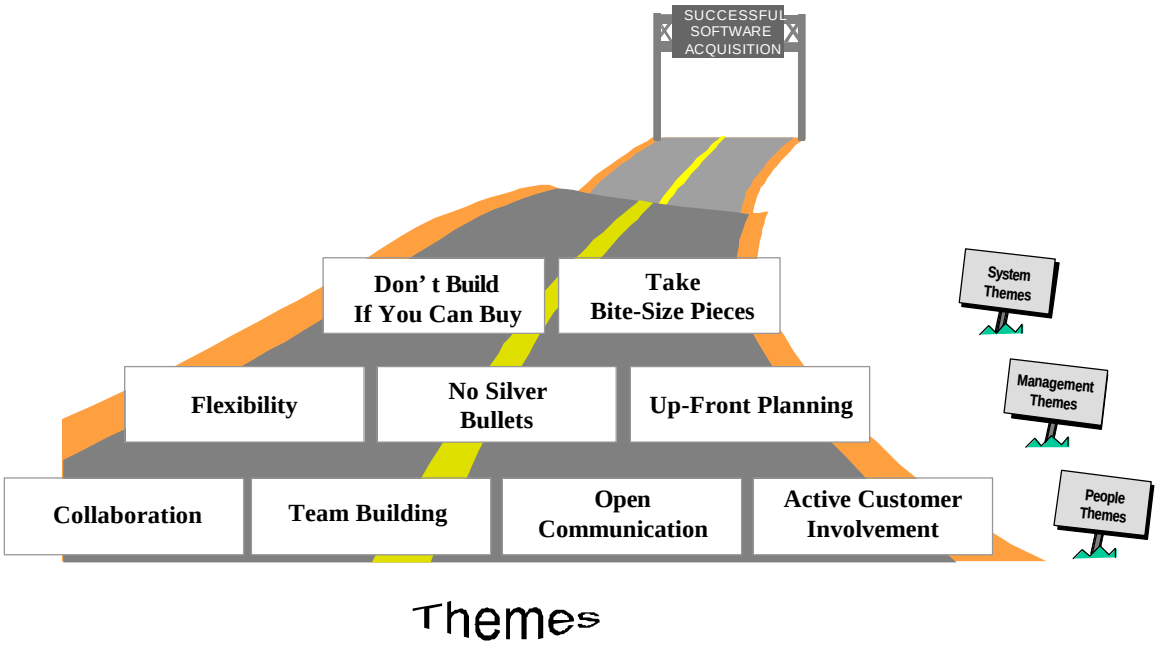


Figure ES-1. Themes On The Road To Successful Software Acquisition

The *people themes* have been likened to *partnering*, whose practice has proven beneficial on construction projects. For transportation agencies that don't build ITS software with in-house staff (the usual case), the customer works together with a contractor to achieve common goals instead of having an adversarial relationship. They continually work at *open communications*, and *collaborate* on all activities, from requirements to risk management to system acceptance. This requires a greater customer role than many are used to; *active customer involvement* is essential. This in turn requires that project managers not go it alone. Instead, they practice *team building*, both within their agency and with the software contractor.

The *management themes* guide the management of an acquisition. *Flexibility* is needed in the contract to accommodate change and take advantage of the opportunities presented by application of the people themes. There must be the recognition that there are *no silver bullets*; no one acquisition practice or contracting mechanism is a panacea that can be relied upon to rescue a project. *Up-front planning* is needed early in the acquisition,

even for activities such as system acceptance that do not take place until late in the acquisition process.

System themes relate directly to the final product. *Never build if you can buy* existing products. Purchasing pre-existing products alleviates many of the risks associated with building custom software. For most types of ITS systems, off-the-shelf products or components are available. Unique requirements can preclude their use, but any such requirements should be examined to determine whether they really are important or whether the system is over-specified. Ask yourself why your requirements are so much different from everyone else's. Many projects fail because they attempt to do too much at once. By *taking bite-size pieces*, an acquisition is more manageable. Contracting mechanisms must be chosen that allow for this instead of those that call for an all-at-once "big bang" approach.

The various software acquisition activities that are built upon these themes will now be discussed.

Acquisition Activities

An early activity in an acquisition is *building a team*. Following are some of the skills that must be tapped (from within your agency, if possible) and included on the team:

- *Software technical experts* assist with requirements, scheduling, costing, technical reviews, and eventually liaison with the software contractor. These individuals are difficult to find, especially for public agencies.
- *End users, maintainers, and system administrators* have very different perspectives on systems than do engineers. Their membership on the team helps ensure that their needs are addressed.
- *Domain experts* ensure that a system addresses operational needs and guide the end users in understanding and operating the system.
- *Contracting and purchasing officials* help select the most appropriate contracting vehicles. A full range of options must be considered as traditional vehicles used on construction, consulting, and other types of transportation projects are not appropriate for software.
- *Software-specific legal staff* assist in resolving intellectual property rights issues to avoid litigation over them.

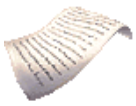
Once the *software contractor* is selected, they become an essential member who must be incorporated into the team.

One ITS manager successfully teamed with his contractor by treating them as part of his staff. They were invited to attend staff meetings and participated in setting milestones for the project.

An initial activity for the assembled team is *project planning*. Write a short project plan. Several parts of this plan are unique to software, or at least more critical for software than they are for other types of projects. These include identification of the following: facilities, acquisition strategy, system environment, risk management, project oversight techniques, end users, acceptance strategy, training concept, and maintenance concept. Clearly, many of these planning activities, especially the acquisition strategy, will take place before the contractor is on board. Writing a plan helps achieve "buy in" for subsequent activities and gives everyone an awareness of the trade-offs that have to be made. Although written during the early part of a project, several sections of the plan address activities that will not take place until late in the life cycle.

Three key activities take place in parallel and feed off one another: developing requirements, making build/buy decisions, and selecting the contracting vehicle.

The first key activity, *developing requirements*, is one of the most important that takes place on a software acquisition. The team members participate in developing a good, sound set of functional and performance requirements. Functional requirements define automated and manual system capabilities. Performance requirements define such items as response time, capacity, reliability, safety, and security. Unlike other transportation projects, software acquisitions should not develop design specs or technical requirements at this stage. (Software *is* different!) The requirements give the *what's* not the *how's*. They address such topics as system functions, response times, reliability, maintainability, security, safety, interfaces, inputs, and outputs. Noticeably missing from this list is detailed human interface requirements on how operators and end users will interact with the system. Rapid prototyping is a better approach for addressing them.



A few ITS examples illustrate what can happen when human interface requirements are specified on paper. In the transit arena, a box on a bus needed multiple keystrokes for a simple function like changing the volume control. This was not apparent from reading the written requirements and was not realized until the box was used operationally. In the traffic arena, incident reports could not be filed until all the fields of an on-line form were filled out. Many of the fields were not particularly important and filling them out delayed the transmission of critical information. But the requirements did not specify the capability to transmit a partially filled out form or allow the ability to retrieve a form and add the missing fields later. Because rapid prototyping techniques had not been used in either case, it was not possible to visualize the implication of the written requirements. Only real-world interactions with the system revealed the flaws that were inherent in the requirements. If you buy existing products, you will at least gain the benefits of someone else's experiences.

In developing requirements, don't ask for too much. Avoid the temptation to "add just one more requirement, it's only a matter of some more software." This keeps the project manageable, minimizes risk, and achieves an operational capability sooner. Use scrub sessions to eliminate inessential requirements. Furthermore, over-specified systems inevitably dictate design and preclude off-the-shelf solutions. Once operational experience is gained with an initial implementation, there's always time to build upon success and add features.

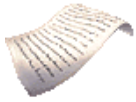
Related to requirements are quality factors. The quality factors address "how well" the system meets requirements and include such "ilities" as reliability, availability, and maintainability. System *flexibility* should also be addressed. Software is inherently flexible. Ironically, software *systems* often are often inflexible; they are not robust to change. You can help achieve more flexibility by asking "what if" questions in regards to future features and what is likely to change. (For example, "What if we added another jurisdiction to a regional ATMS?" "What if the ramp metering algorithms were changed?") Then see if the system can accommodate those changes.

> *Fundamental flaw of software acquisition: "One can specify a satisfactory system in advance, get bids for its construction, have it built, and install it É this assumption is fundamentally wrong. … It is necessary to allow for extensive iteration between the client and the designer as part of the system definition." — [Brooks, 1987]*

At one time it was thought that the key to software success was i) develop a rigorous, complete set of requirements, ii) freeze them for the entire project, and iii) insist that the contractor meet all the *shall's*. Although it would be nice to set aside the requirements and go on to other tasks once the requirements are documented, you unfortunately cannot dismiss them as a "done deal." Unless you go strictly with an off-the-shelf buy (see below), an on-going requirements management process will be needed, carried out collaboratively by customer and contractor. This includes conducting a requirements walk-through with the software contractor.

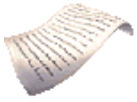
> *"Rule 1 of Systems Integration: The agency and the integrator will never interpret the functional definition in the same manner." — [Phil Tarnoff]*

In a walk-through, every requirement is thoroughly examined until the customer and contractor achieve a common understanding of it. An example of our open communications and collaboration themes, a walk-through also provides another opportunity to scrub requirements and to explore alternatives that replace high risk requirements with lower risk ones. This is true whether you build or buy. If you buy an off-the-shelf product, the supplier is in the best position to identify which modifications are easy and which ones are hard or risky.



One ITS software developer cites the example of an unnamed customer who refused to carry out a requirements walk-through. So the contractor proceeded as best they could in designing the system. Then came the critical design review, a major milestone. But instead of addressing design issues, the review quickly back-tracked to the unaddressed requirements issues. The contractor and customer finally reached a mutual understanding of the requirements, but not without cost. By then, much of the previous design work had to be discarded and re-done. This could have been avoided with a timely walk-through of the requirements.

Once the requirements are revised to reflect the mutual agreements of customer and contractor, they are signed and placed under configuration control ("baselined"). From this point on, changes to requirements are carefully controlled. A careful balancing act that must be practiced. On the one hand, having a stable set of requirements is essential for successful software development. Changing requirements and scope creep can be fatal. On the other hand, "controlled" should not be equated with "frozen". Requirements issues must be addressed as they arise, with all changes agreed to in writing by all parties before they take effect. There should be sufficient teamwork and contractual flexibility to clarify ambiguities, flesh out lower-level requirements not initially addressed, and relax requirements that pose unexpected risk or prove technically difficulty to implement.

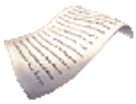


One satisfied customer told his long-time contractor, "The reason we're so successful together is because you always give me 80% of what I ask for."

The requirements become the basis for size, schedule, and cost estimates; build/buy decisions; design and development activities; and acceptance testing. (On too many projects, these other activities are carried out independently of the requirements.) If changes increase the scope of the project, they must be accompanied by schedule and budget relief, or compensated for by eliminating other requirements in the system.

"The most radical possible solution for constructing software is not to construct it at all." — [Brooks, 1987]

The second key activity is *making build/buy decisions*. This decision-making activity is often neglected in spite of the fact that it has the potential of overcoming many of the problems incurred on software acquisitions. Never build the system (or portions of the system) if you can buy it. A matrix showing which vendor products meet which high-level requirements can be used to help you make this decision. Product demonstrations (perhaps at your site) or visits to other sites are some of the ways that will allow you to find out what's available in the marketplace. If no vendor products meet a requirement, carefully consider its necessity and technical risk. Ask yourself whether you are unnecessarily precluding off-the-shelf products. Is a pre-existing, 80% solution good enough?

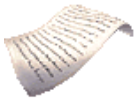


The consideration of the availability of existing products and the willingness to trade off functionality to decrease cost and schedule has been cited as a 'best commercial practice"that is used by the private sector. [Ferguson and DeRiso, 1994.]

For the portions of the system that you buy, only high-level requirements (a features list) may be necessary. For those portions of the system that you decide to have built, a requirements management process such as that described above will be needed.

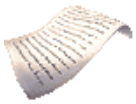
Although buying the system can reduce risk, purchasing software is not a panacea and has its own associated risks. Mitigation strategies are available for addressing them. These include "kicking the tires" or meeting with prior customers before committing to an existing product.

The third key activity is *selecting a contracting vehicle* for the acquisition. Unfortunately, no acquisition vehicles are ideal for software, and the familiar engineer/contractor (design-bid-build) approach is particularly inappropriate.



Engineer/contractor often leads to multiple layers of subcontracting. One ITS software contractor found themselves third tier down on the subcontracting arrangement of a construction contract. They were effectively shut off from all direct contact with the customer. This lack of contact predictably led to a very bad software experience for all parties.

Therefore you will need to work with your contracting and legal representatives on your team to explore the full range of options. Among them are cost-reimbursement, time-and-materials, design/build, design to cost and schedule, and build to budget contracts. Although not often used, time-and-materials contracting offers a number of advantages for software and is allowed under Federal-aid regulations.



A leading-edge traffic management center was successfully built using a time-and-materials contract. A "rolling" development approach was used. The system evolved over time by having new pieces of the system put in place at frequent intervals, typically on the order of several weeks. The contracting approach was credited with being able to adapt quickly to the unforeseen popularity of the Internet when that became a viable vehicle for transmitting traffic information.

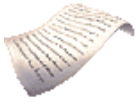
The build/buy decisions influence the selection of a contracting vehicle. In particular, fixed-price contracting (or any type of contract with firm deliverables and fixed ceilings for price and cost) does not provide the needed flexibility for building software, or modifying existing products, although it may be appropriate for some off-the-shelf buys. For fixed-price contracts, requirements will need to be issued as part of the RFP. However, if you decide to go with a cost-plus or time-and-materials contract for building software, then only high-level requirements or a features list need be issued as part of the RFP. For such contracting vehicles, detailed requirements development and requirements management activities are best deferred until after contract award. Regardless of the contracting vehicle that is chosen, it should not be regarded as a substitute for sound management and application of the acquisition themes.

The following paragraphs describe the remaining acquisition activities that have defined beginning and end points. These are followed by descriptions of several on-going activities that take place throughout an acquisition.

The software environment includes interfaces to field equipment, legacy systems, other software products (e.g., operating systems, database management systems),

communications equipment, and systems in neighboring jurisdictions. *Identifying the software environment* is analogous to performing a site survey on civil engineering projects. This gives an overall picture of what the project entails. However, do not unnecessarily constrain the system design or preclude the use of existing products by prematurely specifying computing hardware or operating systems as part of the environment. A product that works well in one location may require significant re-tailoring if it is placed in a different location with a different environment. Choosing a contracting vehicle that results in a low-bid purchase of computing hardware or field devices without regard to the software may inadvertently preclude the use of an existing software product. Or it may result in significant cost and development risk for re-tailoring the software to fit the new environment.

Unfortunately, many acquisitions culminate in conflict and even litigation because of unclear understandings of who has what rights to the software. *Resolving the intellectual property rights* must be done before a contract is signed. Be very explicit and reach detailed agreements with the contractor; a checklist is provided in the body of this document to assist with this.

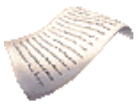


The meaning of the term "software" in the contract language has been a frequent point of contention between customers and software contractors. Often the customer interpreted "software" to include the source code, whereas the contractor meant for "software" to apply to executable or object code.

Even though intellectual property rights issues do not arise until after the system is completed, walk through the checklist with the contractor before a contract is signed. This can be an initial step toward achieving the open communications theme. Procuring the services of an intellectual property rights lawyer who specializes in software has proven to be "money well spent" on several acquisitions.

> *"More software projects have gone awry for lack of calendar time than for all other causes combined." — [Brooks, 1975]*

Two common flaws are common with software *project scheduling*: First, schedules are established independently of the requirements. Second, they are squeezed so tightly that they are set in the impossible-to-do zone, even if everything goes perfectly. Instead, develop a schedule that realistically matches the requirements. More often than not "realistic" equates to "pessimistic". In developing the schedule, use well-defined "yes/no" or "done/not done" milestones.



On one ITS project, milestones were so ill-defined that the participants were openly puzzled as to whether they had met them or not.

In establishing a schedule, get as many independent size estimates for your system as possible, including those of the contractor and the software experts on your team.

" [You] can reduce effort, cost, (and defects) by planning a little longer schedule." — [Putnam and Myers, 1996]

Once a realistic schedule has been drafted, one of the most cost-effective ways of lowering the cost and the total effort on a project is simply stretch out the schedule. Doing so may defy intuition, but is another example of " software is different" .

Once a schedule is set, if the requirements change, make corresponding changes in the schedule to keep the two in agreement. After the software contractor's activities begin, use actual progress to derive more realistic schedule estimates for the remaining activities.

Even though system testing does not take place until after the system is built or bought, plan a formal system *acceptance testing* strategy early, before an RFP is issued. Reflect your approach in the contract. Acceptance test preparations (preparing test cases, setting up a testing environment, etc.) need to begin soon after contract award. Schedule them to take place in parallel with other software activities. This will avoid a common problem of treating acceptance testing as an after-thought.

Acceptance tests are based on the requirements. They should be rigorous; simple benign tests are not sufficient. Tests should include functional tests, maximum capacity and stress tests, erroneous inputs tests, stability tests, and integrity tests. When testing takes place, carry it out as a collaborative teaming activity.

One ITS manager told us that " maintenance kind of caught us by surprise." Don't let that happen to you. Plan for the support activities—*training, operations, and maintenance*— early in the acquisition. Assign contractor responsibilities for support activities in the contract, and allot adequate time to prepare for them. Give serious consideration to including contractor maintenance in these responsibilities as opposed to having maintenance carried out by in-house staff. Adequate resources are also needed: over the life of a system, support activities generally consume more budget resources than it costs to build the system initially.

On-Going Management Activities

We now turn to several on-going activities that take place in parallel throughout an acquisition. We have already addressed one of these—requirements management.

Even after a contract is issued, the customer still has an active role to play. *Project management* activities include project reviews, document reviews, and the use of quantitative measurement data to gain visibility into contractor progress. If schedule slips occurs, do not try to play " catch up" .

Intuition gained from other endeavors for meeting a schedule "more workers, money, overtime, computer time—doesn't seem to work for software." — [Putnam and Myers, 1992]

Either stretch the remaining schedule in accordance with the slip or reduce functionality in the same proportion as the schedule slip. Project management entails not only contract management, but also expectations management of other stakeholders not directly working on the project.

Software configuration management is another essential on-going activity. Baselines are established and serve as a controlled basis for future work. (A baseline is a "snapshot" of everything associated with the software including such items as the source and object code, requirements and other technical documentation, test cases, and problem reports and their status.) Formal procedures are established and followed for making changes to the baseline; otherwise, different aspects of the system will rapidly become "out of synch" with one another. The customer must ensure that the contractor establishes sound configuration management procedures and follows them.

Software risk management is another on-going activity that is carried out throughout the life of a software project. Risk management steps include risk identification, analysis, planning, resolution, and monitoring. Risk management is most effectively done as a teaming activity between customer and contractor since their different perspectives on the system lead them to identify different risks. For risk management to work, there must be an atmosphere that fosters project personnel to come forward with risks without "finger pointing."

Topic Sheets

Several pertinent software topics are addressed in the following topic sheets at the end of Volume II of this document:

- *Rapid prototyping* is the recommended approach for fleshing out human interface requirements.
- *Security* must be built into system from the outset. A number of security mechanisms are available to provide a range of necessary security services.
- The *Software Acquisition Capability Maturity Model* can be used by an agency to assess its readiness to acquire software.
- The *Software Capability Maturity Model* can be used to assess contractor capabilities to develop software.
- *Software Safety* is concerned with ensuring that the software does not cause hazardous, life-threatening, or other highly undesirable conditions to occur.
- The *Year 2000 Problem (Y2K)* and the similar GPS-rollover problem are challenges faced by software acquisitions.

Concluding Remarks

We hope this document will help you with your ITS software acquisition. It starts out by explaining how software is different. The rest of the document is essentially our recommended response to that difference. We have taken a process-oriented approach. It is centered around a series of themes that deal with the system, the management outlook, and most importantly, the people. Then we built upon those themes, showing how they play out in certain key activities.

To be sure, we haven't been able to give you all the answers. Your software acquisition will still be hard work, requiring your hands-on, active management involvement. The road ahead may not be a totally smooth one; there are no silver bullets. But perhaps it will be less bumpy because you'll know the potholes to avoid. And that may help keep small risks from growing into major problems.

As Brooks wrote in his classic paper *No Silver Bullet: Essence and Accidents of Software Engineering*, "There is no easy road, but there is a road."

Executive Summary References

F. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering", *Computer*, vol. 20, pp. 10-19, April 1987. (Also reprinted in *The Mythical Man-Month: Anniversary Edition*, Addison-Wesley, 1995)

F. Brooks, *The Mythical Man-Month: Essays on Software Engineering* , Addison-Wesley, 1975. (Also reprinted in *The Mythical Man-Month: Anniversary Edition*, Addison-Wesley, 1995)

J. Ferguson and M. DeRiso, *Software Acquisition: A Comparison of DoD and Commercial Practices*, Software Engineering Institute Special Report CMU/SEI-94-SR-9, 1994

C. Jones, *Software Project Management: What Works and What Doesn't*, talk at SD '97 Conference, Washington DC, September 29, 1997

S. McConnell, *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, 1996

L. Putnam and W. Myers, *Executive Briefing: Controlling Software Development*, IEEE Computer Society Press, 1996

L. Putnam and W. Myers, *Measures for Excellence: Reliable Software on Time, within Budget*, Yourdon Press, 1992