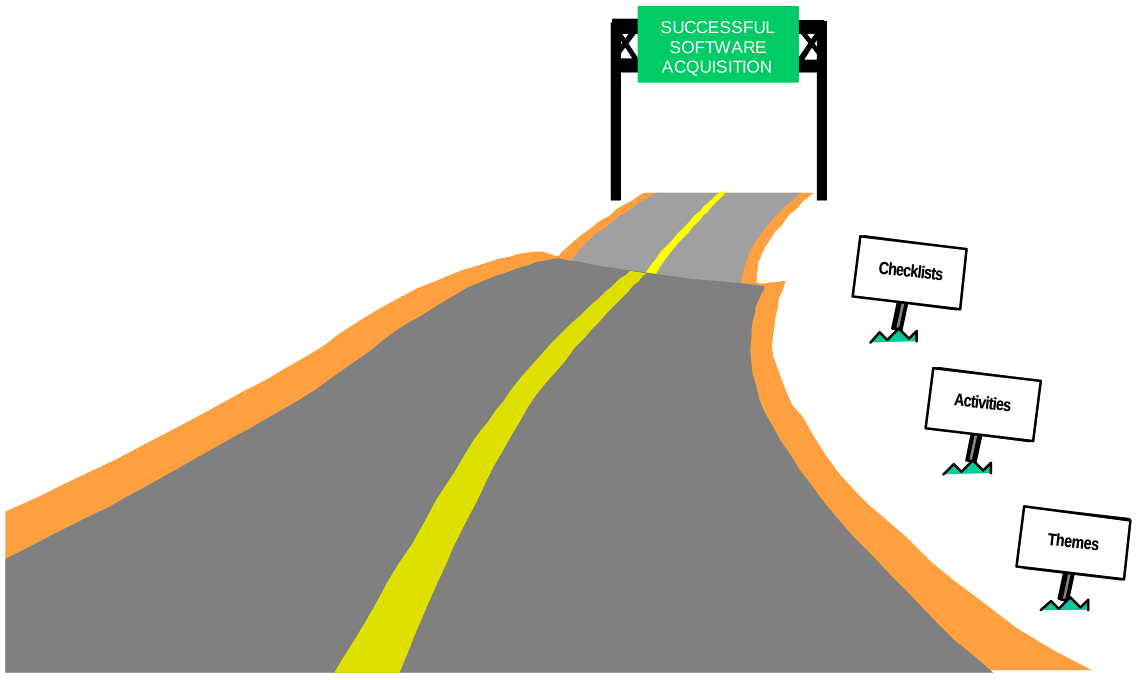
U.S. Department of Transportation

Federal Highway Administration

The Road to Successful ITS Software Acquisition

Volume I: Overview and Themes



The Road to Successful ITS Software Acquisition

Volume I: Overview and Themes

July 1998

Prepared for the Federal Highway Administration
by Mitretek Systems

1. Report No. FHWA-JPO-98-035	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle The Road to Successful ITS Software Acquisition Volume I: Overview and Themes		5. Report Date July, 1998
		6. Performing Organization Code
7. Author(s) Dr. Arthur E. Salwin		8. Performing Organization Report No.
9. Performing Organization Name and Address Mitretek Systems 600 Maryland AVE SW STE 755 Washington, DC 20024		10. Work Unit No.
		11. Contract or Grant No. DTFH61-95-C-00040
12. Sponsoring Agency Name and Address Department of Transportation Federal Highway Administ ration ITS Joint Program Office 400 Seventh ST SW Washington, DC 20590		13. Type of Report and Period Covered
		14. Sponsoring Agency Code HVH-1

15. Supplementary Notes

Bill Jones and Lee Simmons

16. Abstract

This document assembles best practices and presents practical advice on how to acquire the software components of Intelligent Transportation Systems (ITS). The intended audience is the " customers" --project leaders, technical contract managers, decision makers, and consultants--who are responsible for one or more ITS systems.

The document presents a series of "themes" that serve as guiding principles for building a successful acquisition. Included are people themes of collaboration, team building, open communications, and active customer involvement, which have been likened to partnering; management themes of flexibility, "no silver bullets", and up-front planning; and system themes of "Don't build if you can buy" and "Take bite-size pieces". Software acquisition activities that build upon these themes are presented in subsequent chapters. Among the activities covered are building a team, developing requirements, making build/buy decisions, resolving the intellectual property rights, acceptance testing, and project and risk management. Also included are "war stories" to illustrate the various points, as well as key point summaries and checklists to facilitate use of the material. The document concludes with short stand-alone topic sheets that introduce various relevant software topics.

Key Words Software, Acquisition, Procurement, ITS, Intelligent Transportation Systems	18. Distribution Statement No restrictions.

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No of Pages 68	22. Price

Form DOT F 1700.7 **(8-72)** Reproduction of completed page authorized

ACKNOWLEDGMENTS

This document could not have come to be without the contributions of many players. First, thanks are owed to over thirty interviewees from throughout the ITS community. They were promised anonymity so they could freely relate their ITS experiences. And that they certainly did, taking time out from their busy schedules and honestly relating their software experiences. Many of their "war stories" have been disguised and incorporated as side bars in the body of the document. The interviewees also offered valuable guidance on what topic areas to cover, helping us to get off on the right foot.

A review panel served as a surrogate audience for the document. Transit, traffic, and software engineering perspectives, from both the public and private sectors, were represented. The panel members took time from their busy schedules to review drafts of the document, discuss them with us at two review workshops, and provide many valuable comments. In alphabetical order, the panel members were Ron Damer, Gary Euler, Bill Hiller, Les Jacobson, Kay Johnson, John Marciniak, and Jim Wright. Several individuals at PB Farradyne Inc. also reviewed draft material.

The following individuals and organizations wrote white papers and other materials that were heavily borrowed from or incorporated directly into this document: John Marciniak, PB Farradyne Inc., TransCore (Kay Johnson), the Software Engineering Institute (Ron Damer and John Waclo), and Mitretek (Keith Biesecker, Fred Dion, Walt Key, Chris Reedy, and Susan Riegner). These individuals also provided invaluable consultation.

Several Mitretek colleagues deserve special recognition. Tight deadlines would not have been met without Rich Bolczak's invaluable contributions in revising the document. As usual, his sound advice and good humor are most appreciated. Jan Spore produced all the graphics. Her ability to generate quality work at incredible rates of productivity is truly amazing. Greg Hatcher coordinated with outside organizations, including FHWA, to keep the project running smoothly. Donna Dion capably handled the production and distribution of draft and final versions with her usual positive attitude, and along with Cheryl Simpson assisted with the review panel arrangements. Finally, Mary Helen Malow assembled reference material from multiple sources.

Dr. Arthur E. Salwin (Lead Author)
Principal Engineer
Intelligent Transportation Systems
Mitretek Systems

PREFACE

Perhaps you are a project leader, a technical manager for a contract, or a decision maker who is responsible for one or more Intelligent Transportation Systems (ITS). You may work in a state or local department of transportation or in a transit agency. Or you may be a consultant advising one of the above. In any case, this document is intended to help you, the customer, acquire the software components of an ITS system. We do not assume any software background on your part. We recognize you probably won't be acquiring software *per se*. However, if it's an ITS system, it undoubtedly has a significant

In putting together this document, one of the things we wrestled with was the title. Should we use the term "procurement"? "acquisition"? "purchase"? What we found was that the same word has different meanings to different agencies. To some, "procurement" or "procure" means the whole process from project conception to operational use. To others, "procurement" refers to only that part of an acquisition concerning the legal and contractual issues associated with issuing an RFP and awarding a contract. To them, "acquisition" is a more encompassing term, and that is the one we went with in writing this report. By ***acquisition*** *we mean "The process of obtaining a system or software product." [IEEE, 1993, page 5] But whatever term you prefer, we are referring to the entire process of "getting" the software.*

For our purposes, we view the ***customer*** *as being in the public sector. Customers are the personnel -- project managers, end users, contracting officers, and others -- who work in a traffic or transit agency. They issue contracts to acquire software from private sector contractors. This software can be custom-built from scratch, or it can be an existing product that is bought by the public agency. Clearly, other paradigms are possible. (The agency could develop its own software, for example.)*

software component. And software acquisitions often mean trouble.

In preparing this document, we surveyed the software engineering literature and conducted many interviews with public-sector and private-sector personnel who had been involved with ITS projects. Whether they were involved with traffic or transit projects, the interviewees raised the same general issues. Public-sector and private-sector interviewees also tended to raise the same issues, albeit with different perceptions of those issues. Most of the public-sector interviewees indicated that they had experienced troubles of one sort or another with their software or software acquisitions. At the same time, the system suppliers (typically contractors) said that they lose money on ITS software, typically twenty percent of the size of the software contract.

A true lose-lose situation.

Certainly, software problems are not unique to ITS, to transportation, or even to the public sector. In fact, it would be surprising if ITS hadn't experienced them. Furthermore, although most of the interviewees were new to software, the problems they encountered, and the solutions they proffered, were largely in agreement with findings in

the software engineering literature. This tends to reinforce our belief that sound practices, applied successfully elsewhere, are largely applicable to ITS systems.
Accordingly, we have attempted to assemble best practices and present practical advice on how to carry out a software acquisition. This document is not meant as "thou shall comply"; rather it is a collection of helpful ideas. Please do not treat our recommendations as rigid guidelines or standards that you must follow. Choose from them selectively; they may not all be applicable to your particular project. Pick the ones that seem the most useful or beneficial to you. Tailor your approach, but do take advantage of the information presented here to improve your chance of success.

Our scope is the overall system, including the software and the computing platforms on which it resides. Not included are such items as the building facilities or the field sensors. Even though the target audience is the customer—that is, public-sector officials who work in transportation or transit agencies—we also present private-sector perspectives on the issues. We address generic acquisition and management issues and do not focus on the particular functions that a freeway management system or transit vehicle location system should possess. To be sure, we don't have all the answers. Indeed, one of our recurring themes is that there are no silver bullets. Also this is not an academic text book on software or a handbook on how to *develop* it. So we do not include, for example, material on how to code or design a software system.

One senior ITS official noted, "I've never been involved in a software [acquisition for which] I've been truly satisfied at the end." If this document helps enable you to be satisfied with your software acquisition experiences, or improves the success rate of software acquisitions, then we've succeeded in our mission.

DOCUMENT ROADMAP

We recognize that probably no one would ever pick up this document and read it straight through from beginning to end. Therefore, to make it more convenient for the reader, we've divided it into two volumes. The shorter Volume I, *Overview and Themes,* presents the foundation upon which the rest of the document is written. We strongly urge you to read it in its entirety first. A software acquisition should be built around the themes discussed in this volume. The longer Volume II, *Software Acquisition Process Reference Guide,* has the specific activities that build upon these themes. As your acquisition unfolds, various topics discussed in this volume will become relevant to your needs. When they do, you can turn to the appropriate chapters.

CAUTION

Chapters may become relevant sooner than you expect. Many acquisition activities have to be first addressed long before they actually occur. Take acceptance testing, for example. Even though acceptance testing does not take place until after development activities are complete, it must be planned for early. In other words, the chapter on acceptance testing will become relevant long before testing actually takes place.

The chapters in this document are grouped into six parts. Parts One and Two appear in Volume I; Parts Three through Six appear in Volume II.

Volume I

Part One sets the stage for the recommendations that follow later. It begins with a discussion of software in general and shows how software acquisitions are different from other acquisitions that you may be familiar with.

- Chapter 1, *The Nature of Software* explains how "software is different." The purpose is to motivate the reader to be receptive towards new approaches for their software acquisitions.
- Chapter 2, *Software Acquisition In A Larger Context* shows where software fits into the overall system acquisition process.
- Chapter 3, *Differing Perceptions of ITS Software* gets closer to home, showing how the public and private sectors within the ITS community view software acquisitions very differently. Much of the mistrust that arises between the two camps can perhaps be overcome if each learns where the other "is coming from."
- Chapter 4, *Types of ITS Software Systems* is more specific. It presents the range of ITS systems whose acquisition is the subject of this document.

Part Two responds to the differences discussed in Part One.

- Chapter 5, *Themes of Successful Software Acquisition* introduces a number of themes around which successful software acquisitions are built. These themes guide the various acquisition activities and are applied over and over again if your software acquisition is to be successful. Collectively, they represent a different way of doing business; our response to "software is different."

Volume II

Parts Three and Four discuss the various activities that constitute a software acquisition. Each chapter discusses a separate activity. Collectively, these activities encompass the entire period from the system's initial concept, through the software development, on to the end of the system's operational life. The themes introduced in Part Two recur throughout the various activities. When they do, we'll call them out.

The activities in Part Three have defined beginning and end points.

- Chapter 6, *Sequence of Acquisition Activities* gives an overview of these activities and discusses how no single timeline can be generated to describe all acquisitions.

  The next two chapters discuss activities that necessarily take place early in the acquisition.

- Chapter 7, *Building A Team* discusses the players that must be assembled to work together. This must be done early, so they can work together and carry out the acquisition.

- Chapter 8, *Planning The Project* discusses the project plan used to organize an acquisition. Even activities that will not take place until late in the acquisition must be included in this plan. The plan helps to ensure that all the team members are trying to achieve a common goal.

  The next three chapters discuss key activities that drive the rest of the acquisition: These activities all feed off one another and to some extent take place in parallel.

- Chapter 9, *Requirements* is divided into two subchapters: 9A, *Developing Requirements* and 9B, *Requirements Management*. Developing requirements culminates in a requirements document. However, attention to requirements cannot end at that point. Requirements management is still needed for the remainder of the acquisition. Requirements creep must be avoided, but at the same time requirements cannot be "thrown over the fence" and forgotten. Although requirements management is an on-going activity that could have been discussed in Part Four, we chose to include it here to keep all the requirements-related material in one chapter. The length of this chapter reflects the importance of requirements in a software acquisition.

- Chapter 10, *Build/Buy Decision(s)* urges you to give serious consideration to using off-the-shelf systems or system components, rather than building your own. However, this is not a panacea, and the risks in doing so are also discussed.
- Chapter 11, *Selecting the Contracting Vehicle* introduces the various contracting options and discusses their applicability to software acquisitions.
- Chapter 12, *Identifying The Software Environment* discusses the hardware, software, and communications context of the system.
- Chapter 13, *Resolving The Intellectual Property Rights* addresses the contentious issue of who has what rights to the software once it's developed.
- Chapter 14, *Project Scheduling* shows how to pull together the various planned activities into a realistic and achievable schedule.

  The last two chapters in this part address activities that do not take place until the end of the acquisition. Nonetheless, planning and preparation for these activities must begin much earlier.

- Chapter 15, *Acceptance Testing* discusses how to determine whether the system is ready to go operational in a way that is fair to both the customer and the contractor.
- Chapter 16, *Training, Operations, and Software Maintenance* discusses three important activities that collectively will probably take up considerably more than half the budget over the life cycle of the system.

The activities in Part Four take place throughout the entire acquisition. The contractor may do the bulk of the technical work, but the customer still has an active and vital role to play even after contract award. Since the various activities have no natural time sequence to them—they all take place simultaneously—the chapters are ordered alphabetically. As in Part Three, we call out the various themes when they occur.

- Chapter 17, *Project Management* focuses on gaining visibility into the project and what to do if that visibility uncovers a schedule slippage. It also addresses quality management steps that can be carried out to achieve various quality factors such as reliability and maintainability of the system.
- Chapter 18, *Software Configuration Management* discusses configuration management and baselining activities. Without these activities, the various parts of the acquisition can rapidly become out of "synch" with one another.
- Chapter 19, *Software Risk Management*, focuses on how to identify risks and manage them before they become problems.

Part Five wraps things up.

- Chapter 20, *Best Practices Checklist and Key Points Summary* provides a final checklist summarizing best practices. It also collects together the key point summaries and checklists that appear throughout the document.

- Chapter 21, *Where To Get More Help*, suggests outside sources of information.
- Chapter 22, *Concluding Remarks* sets you on the road to your software acquisition.

Part Six contains a series of stand-alone topic sheets. Typically one or two pages in length, these introduce various software topics "offline," without interrupting the main flow of the document.

Throughout the document, checklists supplement the text. In addition the following icons appear throughout:

Themes

The themes icon is used to highlight when an activity or recommendation is a specific instance of one of the acquisition themes introduced in Part Two of Volume I.

Key Points

Chapters end with a bulleted list of key points that summarize the main messages of the chapter.

Sidebars are used to clarify various points or to relate "war stories" on the software acquisition experiences of the ITS community.

The dictionary icon appears when new terminology is defined.

The stack of reference books appears when references are given to the outside literature.

The Road to Successful ITS Software Acquisition

Table of Contents

The Road to Successful ITS Software Acquisition
List of Checklists

Note: All of the checklists also appear together in Chapter 20.

EXECUTIVE SUMMARY

<u>Software is Different</u>

"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.

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.

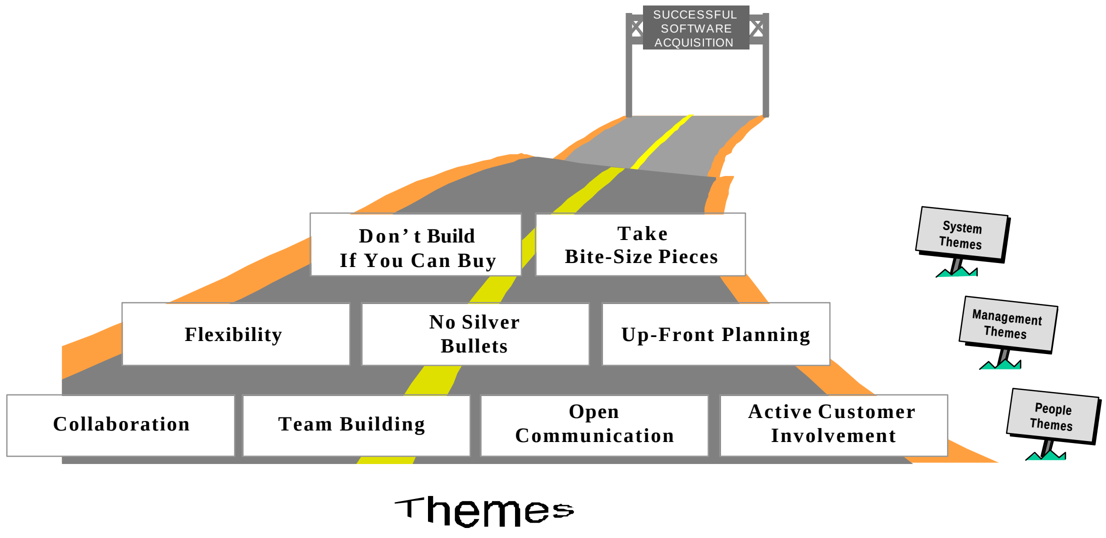


Figure ES-1. Themes On The Road To Successful Software Acquisition

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

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 process.

System themes relate directly to the final product. *Don't 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

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.

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.
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

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.

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.

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…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

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.

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.
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

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."

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.
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

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.]

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?
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.

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.

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.

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.

Unfortunately, no acquisition vehicles are ideal for software, and the familiar engineer/contractor (design-bid-build) approach is particularly inappropriate. 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.

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

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.

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.
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.

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

"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.
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."

PART ONE

SETTING THE STAGE: THE BIG PICTURE

CHAPTER 1
THE NATURE OF SOFTWARE

"Unfortunately, software development does not progress in accordance with the rather simple rules that govern most functions. That is why software projects run beyond delivery dates by many months; overrun budgets, often significantly; and are even canceled about one-quarter of the time. Consequently, you need help to find your way through this thicket." —[Putnam and Myers, 1996]

"Studies have shown that for every six new large-scale software systems that are put into operation, two others are canceled. The average software development project overshoots its schedule by half; larger projects generally do worse. And some three-quarters of all large systems are 'operating failures,' that either do not function as intended or are not used at all … 55 percent of the projects cost more than expected, 68 percent overran their schedules, and 88 percent had to be substantially redesigned." —[Gibbs, 1994]

"The odds of a large [software] project finishing on time are close to zero. The odds of a large project being canceled are an even-money bet."
—[McConnell, 1996, page 81]

Two problems characterize acquisitions that involve a significant amount of software development:

- Products are late.
- Costs overrun the budget.

Not merely risks, these characteristics are near certainties. An unpublished review of seventeen major Department of Defense software contracts showed that none were completed on time. [Humphrey, 1993, page 5] Unfortunately, the difficulties found on software acquisitions are not always apparent at the outset of a project. As Brooks writes in his classic paper *No Silver Bullet: Essence and Accidents of Software Engineering,* "The familiar software project, at least as seen by the nontechnical manager … is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products." [Brooks, 1987]

Why are we emphasizing these problems so early in this document? Because individuals responsible for the software acquisition on an ITS project may not really understand how often these problems do occur. They may carry with them attitudes that reflect their experience with the more familiar civil engineering projects where, according to one interviewee, “overrunning on concrete is considered *really* bad.”

Software engineering is a relatively new field, only a few decades old. But even mature industries have cost and schedule uncertainties when dealing with one-of-a-kind products. Consider the following newspaper article quoting the owner of an arena under construction: ‘[It] will open ‘sometime in the fall.’ As for a date, ‘we don’t know yet.’ ’” [Washington Post, April 24, 1997, page C11.] Or a report addressing a 42 percent increase in school construction costs and cost overruns ranging from 21 to 36 percent. [Rockville Gazette, March 26, 1997, page A-1.] These two examples are taken from the building industry, which dates back at least to the days of the pyramids. So is it too surprising that one-time custom software developments should experience similar troubles?

So why do software acquisitions so often fail? There a number of factors:[1]

- Software systems are generally more complex than other types of systems. Like roads and structures, they have static characteristics, such as the interfaces between systems. But they also have temporal characteristics, such as sensor data inputs or operator interactions, with many subtle timing interactions. Furthermore, systems integration, the most complex and difficult aspect of an overall system, is primarily a software concern.
- Human interfaces—whether they be report layouts on paper or real-time user/operator interactions at a terminal—are a common target of user complaints. Even though the complaints are often based on the operator’s subjective criteria, responding to the complaints necessitates changes. These interfaces are generally implemented in software.
- The ratio of design costs to production costs is the reverse from that found on hardware acquisitions or on construction projects. On construction projects, an architecture design and blueprints are relatively inexpensive when compared with the actual fabrication costs. This serves as a natural impediment to making unrestricted changes to the finished product. It may be years before a new lane is added to a highway, or an addition is placed on a structure. “The high cost of change dampens the whim of [would be] changers.” [Brooks, 1987, page 10] However, on software projects, production costs are relatively inexpensive (e.g., copying a program onto a diskette). This gives the *illusion* that changes are relatively easy to make. So changes are made over and over again.
- Monitoring progress on software projects is difficult. Progress measures that work elsewhere don’t seem to work with software. “It’s not like seeing asphalt being laid down.”

[1]Taken in part from [Humphrey, 1993]

- Gaining visibility into the software is difficult. Managers complain that paper documents and deliverables provide little insight. "It was difficult to get a sense of the look and feel of the eventual system."
- Even after a software system has been successfully implemented, there can be differences. Traditional capital investments made by DOTs—from telephones to street lamps—have typical life expectancies of twenty years or more. However, software systems may have a life expectancy of as little as five years.

In short, across multiple disciplines, many have come to realize that, in general

"Software is Different"

ITS software appears to be no exception to this general observation. Transportation and transit officials who have acquired ITS systems have come to the same conclusion. One interviewee from the ITS community said it best: "Software is a very different animal." What can be done to accommodate the differences?

The Bad News: Obvious approaches that don't work

Let us consider some possibilities.

Suppose you hire an experienced manager with a good track record on other types of transportation projects. Unfortunately, the schedule and budget woes inherent in software are different from the experiences gained on more familiar civil engineering projects. The successful experiences elsewhere can lead the manager to false expectations for their first software project. As the project progresses, this is compounded as the managers struggle to find an effective means to monitor progress. Measures that had been successfully used in the past may prove to be grossly inaccurate for software. [Putnam and Myers, 1996, page iv] In addition, acquisition practices that are successful on materials contracts (e.g., fixed-cost contracts) don't necessarily work with software. In short, the intuitions gained from managing other types of projects don't necessarily apply to software.

How about hiring a manager with a little software experience? Perhaps one who has taken a course on computer programming. Actually that may do more harm than good. It turns out that the intuitions and insight gained on small software projects are misleading and don't scale up on real-world systems.

Perhaps an analogy to a familiar real-world problem will give an indication as to why intuition gained on small scale problems simply does not scale up. Suppose you are about to embark on a car trip across the country, but your only travel experience has been on short, local trips of a mile or less. What have you learned from them? Things like getting the kids buckled in, adjusting your mirrors, and warming up the engine are big deals and the most time-consuming parts of a trip. Once those tasks are carried out, driving to your destination is relatively quick. However, these lessons have not prepared you for the cross-country trek, where obtaining maps, planning routes, making reservations, finding restaurants, and keeping the gas tank filled are the challenges. The small trip experiences are still there at the start of the trip, but are dwarfed by the much longer time spent driving across the country,

In short, the quick and easy approaches won’t suffice.

The Good News: The differences can be accommodated

Since software is different, it must be acquired and managed differently. Traditional approaches, which are oriented towards civil engineering projects and focus on production, must be adjusted to accommodate the nature of software. There are established techniques that do work. We’ll cover some of them later on in this document. For example, we’ll introduce techniques that overcome the problems associated with gaining visibility into a software project.

In contrast to the pessimistic experiences cited in the quotes at the beginning of this chapter, 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]

Key Points

- *Software acquisitions are different from other types of projects.*
- *Missed schedules, cost overruns, and lack of visibility into the software and software development are common.*
- *Different approaches are therefore needed to manage software projects.*
- *There are established managerial techniques that can be relied upon to overcome the problems.*
- *ITS software experiences are similar to those encountered on other types of software projects.*
- *This document is intended to help you find your way through what has been termed the “software thicket.”*

CHAPTER 2
SOFTWARE ACQUISITION IN A LARGER CONTEXT

Software is seldom acquired for its own sake. It is generally acquired as part of a system acquisition in the context of an overall acquisition process.

Figure 2-1 is a high level overview of this context.

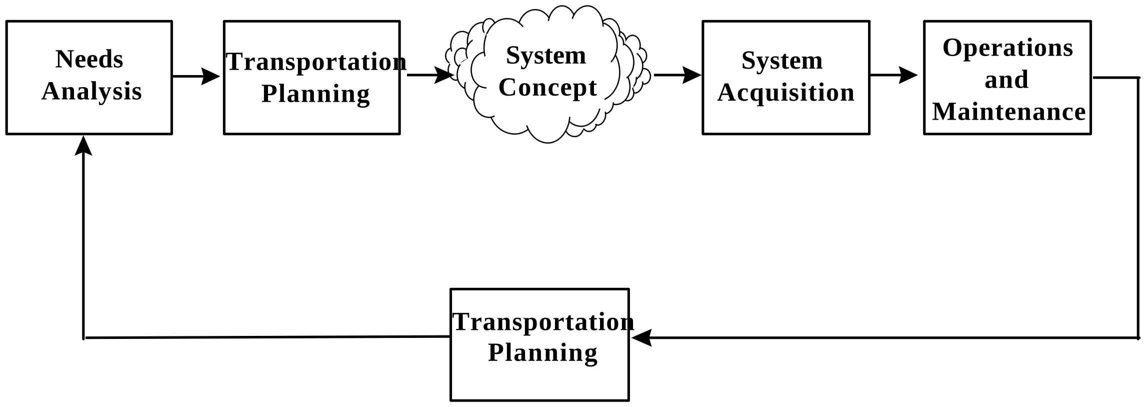


Figure 2-1. Overall Context For A Software Acquisition

The process begins with a *needs analysis,* which identifies transportation needs or problems.

Transportation Planning proposes solutions to these problems. It entails regional planning activities, and the development of a Transportation Improvement Program or TIP. The TIP identifies particular systems, perhaps including an ITS system that you will be responsible for. Use the National ITS Architecture as one of your inputs in generating the TIP.

With project go-ahead, the *system concept* drives your acquisition. Again, the National ITS Architecture can be used, this time to help in defining your concept. At this early stage of a project, the concept may only be a vision in your head of where the project is headed. Or it could be a formal document, setting forth system goals and objectives. Whatever form it takes, it serves as your starting point, showing the direction in which you are headed. Projects without a vision can easily get lost along the way. They may result in a collection of components that work, but do not produce intended results. Or they can get completely derailed and never produce a working product.

The *system acquisition* turns the concept into reality. Included in a system acquisition are software acquisition activities, the subject of this document, discussed in parts three and four. If the acquisition is successful, the system will become operational. *Evaluations* of the operational system provide feedback that is used to drive further needs analysis. At the same time that *operations* take place, there are *maintenance* activities. There are both hardware and software maintenance activities. In fact, over the life of a system, the total effort, and cost, devoted to software maintenance often dwarfs the amount spent on software development activities during the acquisition phase.

Let's take a closer look at the System Acquisition box. Figure 2-2 expands Systems Acquisition into a number of activities, encompassing both hardware and software.

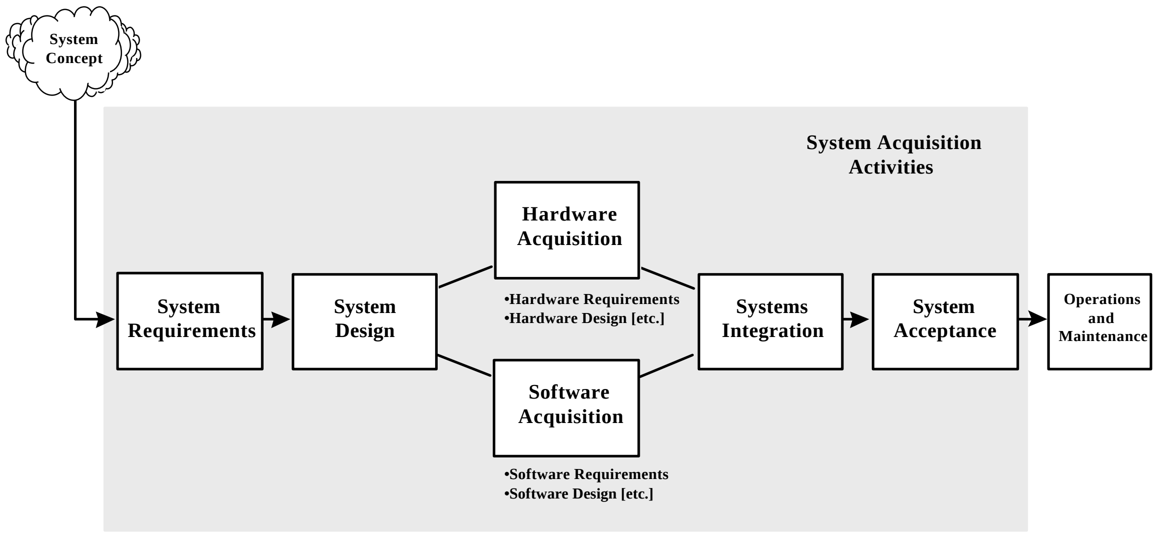


Figure 2-2. System Acquisition Activities

The system design allocates functions between the two. This is the basis for the software requirements, which in turn drive the software design and subsequent software acquisition activities.

As shown in the figure, the hardware and software must be integrated before a system can be accepted and used operationally. Let us stress that figure 2-2 is a conceptual viewpoint only. Not every activity would necessarily take place as a distinct step on every project. Furthermore, system requirements, system design, and software requirements usually don't proceed as a series of sequential steps, each one waiting for the completion of the previous one.

Key Point

- *Although we will focus our attention on the software acquisition activities, they take place in the context of a system acquisition, which in turn is part of an overall process.*

CHAPTER 3
DIFFERING PERCEPTIONS OF ITS SOFTWARE

"The perception is more important than the reality."

As we interviewed ITS personnel for this document, it became apparent that the public sector and private sector have very different perceptions of the software acquisition process. Since these perceptions drive many of their decisions, it is important for each sector to know where the other is "coming from."

Before we began writing this document, we interviewed over thirty individuals with ITS software experience in traffic and transit management. The interviews were equally divided between public-sector (the customer) and private-sector (the contractor) individuals. We wanted to know what types of information would be useful to the practitioner. The interviews also provided us with the opportunity to capture 'war stories" that are sprinkled throughout this document as sidebars to the main text.

Interviewees were promised anonymity so they could speak freely. In spite of their busy schedules, almost without exception the interviewees willingly gave of their time and offered additional help if needed. Even though we cannot acknowledge them by name, we are very grateful to them.

The following paragraphs summarize comments of the public-sector and private-sector interviewees on a number of issues. The perceptions presented are those that were typically expressed. However, it is recognized that any given individual may not hold all the views indicated for their corresponding sector.

Not Getting "Locked In" — Public-Sector Perception

The most common viewpoint expressed by the public sector is the importance of not getting locked into a single contractor. When they do get locked in, they lose all leverage in negotiating follow-on work. Then they get burned by contractors who, they perceive, charge exorbitant prices for, what the public sector perceives to be, trivial changes to the software. Even worse is what happens when a contractor goes out of business or abandons transportation work. Then the customer is left with unsupported and unsupportable software. Several public-sector interviewees cited examples of this happening. To avoid getting locked-in, public-sector customers often require that source code be delivered along with the rest of the system when the project is complete.

Not Getting "Locked In" — Private-Sector Perception

The private sector perception is that getting access to source code is not the answer for not getting locked in. It just adds to the cost. After paying extra for a license to the source code, customers find that it does not prove useful to them. The software is too complicated for an outside party to maintain or modify it. Many seemingly trivial changes to software can in fact subtly impact hundreds of lines of code or be very difficult to implement. Even if the customer succeeds in using the source code to make changes to the software, there may be several undesirable consequences. The changes may negate any warranties associated with the system. For off-the-shelf systems, the changes will be lost if the system is upgraded to the next version of the vendor’s product. The customer thus faces two undesirable options: i) either upgrade and re-implement the changes, or ii) get locked into what soon becomes an obsolete version of the software. The latter option precludes maintenance support since vendors cannot simultaneously maintain all previous versions of their product line.

In at least one case, customer insistence on having access to source code resulted in the contractor re-developing existing software from scratch. They delivered this custom version to the customer rather than releasing their proprietary product. This is an extreme example of how gaining access to the source code resulted in increased costs. It probably also resulted in a less reliable system.

Customization — Public-Sector Perception

The public sector cites several reasons for customization. First, they need to ensure that the system will meet their needs. The perceive that their location has truly unique requirements that must be met. In addition, the public sector may not agree that the algorithms in an existing package are optimal for their application. Second, there are certain established procedures that their end users are already familiar with. The end users cannot be expected to relearn their jobs to accommodate a piece of software. Third, the opportunity to acquire a system does not come along very often. Funding is scarce. So once they get the opportunity, they want to take best advantage of it and not acquire a system that will become obsolete in a few years, even if that means adding custom features. Fourth, since few large, complex ITS systems are purchased nationwide, there may be no acceptable off-the-shelf packages with the needed level of integration or innovation for use region-wide. Finally, custom-engineered solutions are the norm on other transportation projects. Agencies don’t buy off-the-shelf bridges, for example.

Customization — Private-Sector Perception

The problem most commonly cited by almost all the private-sector interviewees is the high degree of customization that the public sector insists upon in their systems. The private sector recognizes that any given site may have some truly unique needs. But they perceive that most customization is unnecessary. It results from their customers trying to engineer the system instead of establishing requirements and then finding the system that provides the best overall solution. When a customer designs a system, it almost always precludes an off-the-shelf solution. The private sector complains that such behavior would not be tolerated in any other field. They complain that for virtually anything else, we buy the off-the-shelf products that best meet our needs. When we buy an automobile we trade off luxury for price, but we don’t go back to the manufacturers and insist that they re-engineer their products. So why, the vendors ask, shouldn't the same philosophy apply to ITS software? One cited the example of buying a toaster—you shop around and find the best one. You may not get a perfect match to what you're seeking, but you would never go back to the manufacturer and tell them to re-engineer their product by putting the knobs on the left side rather than on the right side. But in software such changes seem to be the norm.

Customization leads to unique software being developed for each customer. One vendor complains that they have installed their system in multiple cities, all the installations do essentially the same thing, yet no two copies are identical. This also adds unnecessary risk and reduces system reliability. The contractor's basic system may be a mature product, well tested and operational in many locations. Most of the problems have been exposed and weeded out over time. But new custom-developed software can never have the same assurance of reliability.

One vendor notes that insistence on too much customization precludes customer access to qualified contractors. This vendor has established expertise in a sector of ITS. Since they have a *bona fide* product to sell, they are not interested in custom development software jobs. There's too much risk and such jobs aren't profitable in any case. If they accepted custom developments, they view the risk of having dissatisfied customers as threatening their good reputation. Consequently companies, whom they view as having no particular expertise in their area, win the contracts. They perform poorly, which further feeds the cycle of mistrust.

Customization — Private Sector's Recommended Alternative Approach

The private sector suggests that it would be to everyone's advantage to have vendors resell existing software since this would amortize (spread out) development costs over multiple buyers. (See *The Nature of Software,* Chapter 1 where it is noted that development costs and not production drive the costs of software.) Under the current approach, customers repay for development over and over again. One vendor wrote, "More widespread use of [existing products] would foster more competition and useful innovation. Right now, software providers are forced to consume resources just keeping up with the customization required by different customers, which may or may not appear in the next spec." In other words, they feel that under the proposed alternative, customers would pay less and get more as the state-of-the-art would be given a chance to advance. Transit vehicle tracking and traffic management are examples of systems where vendors claim that satisfactory products are available, but customers insist upon unnecessary customization.

Being Taken Advantage Of — Public-Sector Perception

Public-sector interviewees feel that the private sector takes advantage of the contracting environment. They perceive that high prices are charged for fairly trivial changes to the software that the public sector feels they should have been entitled to anyway. They argue that even if a change was not explicitly called out in the contract, they should not have to pay for its full development cost since the contractor can resell the change many times to other customers. After systems are accepted, they sometimes fail totally and the customer has no choice but to call the contractors for assistance. To be sure, this may happen beyond the end of a formal contract, but a failed system is not what they bargained for.

Being Taken Advantage Of — Private-Sector Perception

Private-sector interviewees also feel that the other side (in this case the public sector) takes advantage of the contracting environment. The private sector perceives that the public sector has no incentive to accept a system since this would mean the end of the contract support. The private sector states that contractors are not getting rich off the software. In fact, they lose money on contracts as the public sector continually requests more changes before accepting a system, without providing additional funds. The private sector says that expecting a contractor to make in-kind contributions is unrealistic: customers should understand that private firms are in the business to make a profit, not as a public service. Vendors complain that in the case of ITS software, they are not contributing excess profit and do not later recover their costs; instead, they swallow the costs and forfeit the ability to even recover their own investments. Indeed in-kind contributions made earlier in the ITS program (on operational tests or early deployment studies, for example) were justified on the basis that those investments would pay off during deployments. The private sector also complains that even after system acceptance, customers call and expect additional support to maintain the system. Contractors have no contractual obligation to provide this support but often must do so to maintain good working relationships.

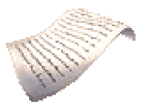
How this document can help

Clearly the differing perceptions do not reflect a healthy situation. Each sector can cite examples of how they were "burned" by the other. Recommendations in this document are intended to overcome some of the problems and help achieve a win-win situation. In particular,

- By *buying* existing products much of the risk and mistrust that result from custom build acquisitions would be avoided. *Team Building* would also help in this regard.
- With *open communications* contractors would better understand customer needs and realize that not all the customization requirements are arbitrary. At the same time, customers would better understand the implications of some of their requirements on the level of effort required, and they would scale back on how much customization is requested.
- Formal *acceptance test* plans would clarify what criteria are used to accept a system. This allows contracts to reach closure and the contractor to get paid. At

the same time, it gives greater assurance to the customer that the system meets requirements and provides sufficient reliability.

- Attention to *maintenance and training* would clarify expected roles during these phases of a project. Provisions for contractor maintenance would allow bugs to be fixed. Documentation addressing system administration functions and interfaces would help alleviate customer dependence on the contractor, who doesn't want to be called upon in any case once a contract has finished.



During the clarification of the maintenance and training roles, mutually beneficial solutions may be discovered. For example, by delivering Graphical User Interface (GUI) development tools, like "report formatters", as part of the contract, many of the customization requirements levied on the contractors can be eliminated. Report formatters are software packages that allow customers to make changes to output reports and screen formats to meet their own needs. If the customer has staff trained in their use, the customers would have less reluctance to formally accept a system, since they would be empowered to make the changes they need. Also they would not need to come back to the contractor so often for support after the system is accepted.

- Contracting language addressing *intellectual property right*s issues may alleviate the need for having source code delivered as part of the contract.

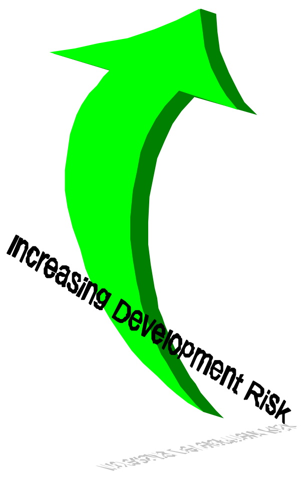
Further details on these topics will be found in the coming chapters of this document.

- *The public and private sectors have very different perceptions of software. These differences manifest themselves in the ways that they approach software acquisitions and each other.*
- *Each sector perceives the situation as lose-win: they lose while the other sector wins. In fact, it's lose-lose; both sectors lose.*

CHAPTER 4
TYPES OF ITS SOFTWARE SYSTEMS

For the most part, ITS project managers are not interested in acquiring software for its own sake. Rather, they are involved in acquiring systems, of which software is a key component. And even when it is not the cost driver of a system, the software often drives the schedule and is the key determiner of system functionality, usability, and reliability. Software is the "glue" that holds the rest of the components together and makes the system work.

What types of systems are we talking about? The figure below ranks various ITS systems in order of increasing software development risk. What these systems have in common is their demanding real-time constraints, which differentiate them from information technology software. Depending upon the function, they must be able to respond within minutes or even seconds to their inputs. They must be extremely reliable, running around the clock without interruption. This contrasts with desktop applications, where system "crashes" are not uncommon, and rebooting the software several times a day may be necessary. Meeting these real-time requirements adds complexity and cost to the software (and to the hardware on which it operates). That, along with the relatively small customer base, explains why these types of ITS software will never be available "shrink wrapped" at $99.



- Regional Advanced Traffic Management Systems
- Traveler Information Systems
- Freeway Management Systems
- Traffic Management Systems
- Transit Management Systems
- Large Signal Systems
- Small Signal Systems

Figure 4-1. Ranking of Various ITS Systems

What is striking from the figure, is that the systems are also ordered by the maturity of the technology, the number of installed systems, and the availability of off-the-shelf product offerings. For most of these systems, off-the-shelf product offerings are available, often from several vendors. The systems at the bottom of the list represent mature technology that can often be bought as commodity products. As one proceeds towards the middle of the list, off-the-shelf products exist in the form of system components that must be integrated. For example, for Freeway Management Systems, off-the-shelf components exist for variable message sign control, but not for traffic-adaptive ramp metering. The Regional Advanced Traffic Management Systems at the top are less mature and represent more developmental risk in regards to software. R&D may be needed on the traffic algorithms in them as well. This imposes additional risk, beyond that inherent in the software. Unfortunately for these systems, there are relatively few or no existing commercial products that could be bought off-the-shelf to mitigate the developmental risk. Customers need to consider whether they have the expertise to acquire such systems.

For the most part the software that integrates one type of system with another (traffic management with transit management, for example), or software that integrates peer systems across jurisdictional boundaries must be custom built and poses development risk.

While this document does not explicitly consider the software acquisition needs of electronic toll collection systems, in-vehicle systems, or systems for Commercial Vehicle Operations, undoubtedly many of the recommendations pertain to them as well. (And, we can speculate, probably even to systems that have nothing whatsoever to do with ITS.)

PART TWO:

THEMES ON THE ROAD TO SUCCESSFUL ITS SOFTWARE ACQUISITION

CHAPTER 5
THEMES OF SUCCESSFUL SOFTWARE ACQUISITION

Dictionary definitions of ***theme****:*
"a recurring unifying subject or idea" [Webster's];
"a melody forming the basis of a set of variations" [American Heritage]

In Part One, we discussed some of the problems that commonly occur on software acquisitions. To address these problems, this chapter introduces several themes around which a software acquisition should be built. Collectively, they constitute a different way of approaching an acquisition, in response to the fact that "software is different." To be sure, not all of the themes are totally new or unique to software. Some represent good management practices that are touted elsewhere, the difference being that, while not doing them on other projects may be acceptable, not doing them for software is fatal. Some of the themes are different, not in kind, but in the rigorous degree to which they must be applied for software. And still others are unique to software and not commonly found on other types of acquisitions, at least not those in the transportation community.

Figure 5-1 illustrates our themes. Let us now consider each of them in turn.

People themes

Collaboration. A software acquisition is a collaborative process. A project manager cannot do it alone or make unilateral decisions. Instead, you must work closely with others in your agency. For example, the end users must be involved at all points to help decide what the system should do, determine how users will interact with the system, and participate in making tradeoffs between cost and functionality. Collaboration also extends beyond organizational boundaries to involve others, especially the software contractor. Only the contractor has the ability to determine the possible design ramifications of seemingly innocent requirements. The contractor also has experience for you to draw upon in determining how best to meet the customer needs.

You can't collaborate unless there's someone to collaborate with. So that brings us to our next theme: Team Building.

Team Building. Many skills are needed for carrying out a software acquisition. No one individual or agency can possibly have all of them. Therefore, a team of professionals is needed. Some of the diverse skills represented on the team include hardware, software, and systems engineering; contracting, operational, domain, and legal expertise. By having your contracting office on the team, you will be able to explore the range of contracting mechanisms and work with them to find the one

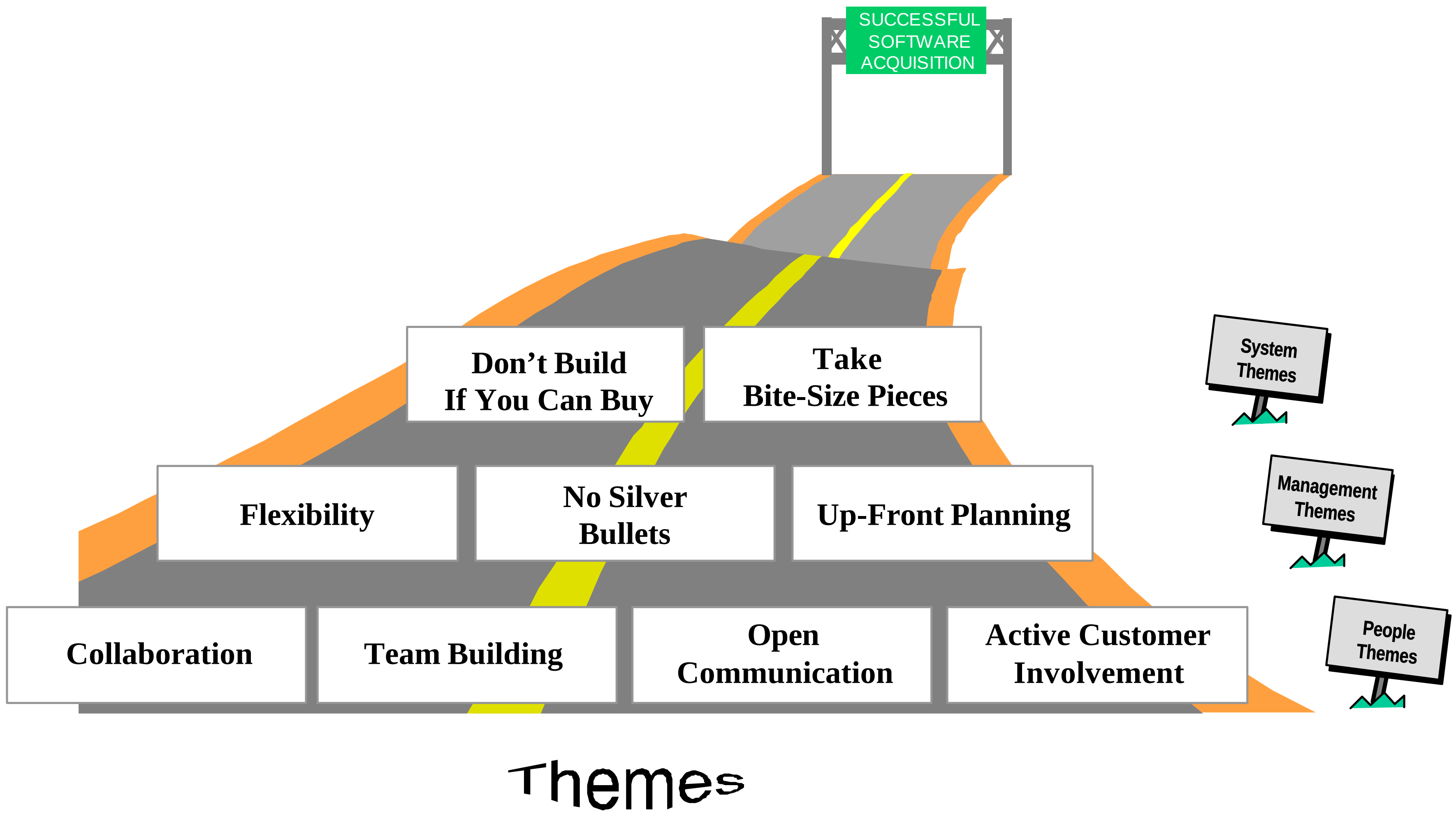


Figure 5-1. Themes on the Road to Successful Software Acquisition

that is most appropriate for software. The various members of the team bring not only different skills but also different perspectives to a problem. For example, the end users on your team will be watching for operational issues that impact their ability to use the system. The team may include other departments in your agency or even other agencies in your region. Peer agencies can be used to provide advice. A most important member of your team will be the software contractor. Even risk management is a teaming activity and should not take place solely on the customer side. The customer and supplier cannot be working at cross purposes; it is essential that they work together to achieve common objectives. If a win-lose mindset develops to "hold the contractor's feet to the fire" or to squeeze the contractor to get something for nothing, that is not teaming. The result is not the hoped-for win-lose, but inevitably lose-lose. Instead, you must strive to achieve a win-win situation.

A team is more than a collection of players with diverse skills. To be a team, the member must work together towards common goals and objectives. Potential bureaucratic obstacles to acquiring a system—whether they be end users, contracting officials, or whoever—must be won over and share in the goal of fielding the system. All must share in the benefits of a successful acquisition. This team building will require maximum use of your political and management skills. The goal sought after is mutual trust.

Team building requires constant nurturing. A key practice to foster team building is the maintenance of continual open communications, our next theme.

Open Communications. Throughout a software acquisition, there needs to be open communications. Open communications with the contractor are especially important. Because customers and suppliers approach software from very different perspectives, there will be misunderstandings unless open communications are continually worked at. The communications must start even before a contract is signed in regards to terms and conditions, especially in regards to intellectual property rights. Open communications proceed with discussions on requirements and continue with any and all decisions on through acceptance testing and maintenance. Throughout the acquisition, each side needs to be able to bring bad news to the other party without fear of being "shot down" or facing recrimination.

The contract must allow for open communications to take place, even if the software is developed by a subcontractor. Requiring a formal communications process for every customer/contractor interaction will only serve to hinder effective communications. But even during formal reviews, open communications must be encouraged.

One successful ITS manager emphasized the need for the contractor to bring forth problems as soon as they arise. "If it's [messed] up, tell me it's [messed] up, and I'll work with you." And, he did. (The exact opposite of a "shooting the messenger" approach.) But he also let them know that if they delayed in bringing forth problems, then the problems were theirs and they would have to absorb any resultant cost overruns.

Another ITS manager cites an example of the misunderstandings that can result when there are not open communications. The contract had required the software vendor to deliver documentation describing the system. The manager was expecting a list of modules, with descriptions of their functions, interfaces, etc. The vendor, reading the contract language differently, supplied an equipment list.

Active Customer Involvement. Collaboration, team building, and open communications all require continuous, active involvement by the customer. You cannot simply "turn things over" to a contractor or systems manager. On other types of transportation projects, it may be sufficient for the customer to take a more passive role, perhaps conducting inspections. But for software, up to half the total requirements and design effort may actually be expended by the customer and end users, even after a software contract has been issued. Clearly, sufficient resources must be allocated for this customer involvement to take place. Active customer involvement also means a willingness to decide upon and commit (in writing, as appropriate) to a specific course of action after open communications have aired the various options.

*Seveal readers have noted the similarily between the people themes discussed here and **partnering**, which is practiced on construction projects.*

Management themes

Flexibility. Construction projects are successfully built to a rigid set of design specifications, at the bid price and for a profit. With software there needs to be more give and take. Flexibility is needed throughout the acquisition: in the requirements, in the working relationships, in the contracting mechanism. But most importantly, there needs to be flexibility in the mindset of those involved. The recognition, as one interviewee said, that "there will be many changes." (Even on construction projects, one expects changes between the initial concept sketches and the final "as built" specifications. As we noted in part one, these design phase activities are more akin to software development than is the actual building of a road or structure.) At the outset of a project, a flexible mindset allows a whole range of options to be considered in selecting the most appropriate contracting mechanism, perhaps even some that have not been tried previously. Similarly, in developing the requirements for a system, there has to be the flexible mindset that the users may not be able to get everything they want; there have to be tradeoffs between functionality and price and schedule.

Requirements will evolve over time. Typically, two percent of the requirements will change per month. [Jones, 1997] The idea that software requirements can be developed, fixed for all time, and then "thrown over the fence" will not work. The implications of certain requirements may not be apparent at first. Only after the acquisition proceeds will it become apparent that they have unacceptable operational implications. Sometimes a seemingly innocent requirement, one that may not even be that important, turns out to have significant impact on the design, imposing significant technical, cost, or schedule risk. The acquisition must be structured in such a way that there is the flexibility to change or remove such requirements. (Identifying them requires open communications.) This is especially true in regards to requirements for the use of specific hardware or software products that may become obsolete by the time they are needed on the project. As tradeoffs are made in cost, schedule, and functionality, it may turn out that a partial solution is the best one. The system may not be able to achieve all that was desired, at least not on the first iteration. Meeting only 80 or 90 percent of the perceived needs may be the most realistic and cost-effective solution. Insisting on more may preclude your buying a rather satisfactory, pre-existing system and instead throw you into a lengthy, risky software development process. "Rigid specifications almost without exception require that systems be custom developed." [Cappelletti and Gerdes, 1994, page 12]

For there to be flexibility in the technical arena, the contracting mechanism must accommodate it and allow change to take place. (And that can only happen if you have already teamed with the contracting office, and initiated open communications with them.) For example, you can encourage innovation by allowing contractors to propose deviations to the requirements. "Best value" procurements enable these proposed deviations to requirements to be accommodated in the contract.

There also needs to be flexibility with respect to cost. For software, it is generally impossible to provide precise, reliable cost estimates at the beginning of a project; only a range is possible. When unforeseen difficulties arise, there must be the flexibility to trade off costs with schedule and functionality as the acquisition unfolds. This may require adding resources through the use of contingency funds (more cost, same functionality) or re-allocating resources (same cost, less functionality). Novel approaches that re-scope the project at each stage of the acquisition process may have to be considered. This would involve the flexibility to have loosely defined contract options that are more precisely defined as the project proceeds.

However, as will be discussed below, too much flexibility is not good either. Requirements creep, in which new requirements are continually added or the scope of the project is increased, must be avoided. One noted software consultant cites stable requirements as one of the two key practices found on successful software acquisitions. (A realistic schedule is the other one.)

No Silver Bullets. As software projects run into trouble, people naturally look for that magic potion that will cure all their software ills. Perhaps it involves placing faith in a particular management practice, a new software tool, a programming language, or a development methodology. Also termed “silver bullets,” the cure-alls unfortunately don’t exist. For example, some in the ITS community feel that new contracting mechanisms are needed for software. This is probably a good idea, since the traditional ones were developed without software in mind. But if and when new contracting options become available, they will not be panaceas. Software issues regarding requirements, costing, testing, and quality control, will still need to be addressed. Successful software acquisition requires the use of multiple sound management and technical practices. Each helps incrementally, but none does the whole trick.

Brooks wrote a classic essay on this theme entitled “No Silver Bullets: Essence and Accidents of Software Engineering.” It is highly readable and is “must” reading for anyone embarking on a software acquisition. [See Brooks, 1987.]

Up-Front Planning. Various acquisition activities such as requirements walk-throughs, design reviews, training, acceptance testing, and maintenance do not take place until after a contract is awarded. In fact, some of these do not take place until very late in the acquisition. Nonetheless you must plan for them up front, before an RFP is issued. This allows the project schedule to call out the various activities and allocate time to them. It also enables the RFP and contract to address the operations and maintenance concept, system acceptance criteria, and the intellectual property rights to the software.

In the technical arena, planning is needed up front to address such issues as software safety, open systems, and security. Such features must be designed into the system from the start; they cannot be added later, at least not without considerable cost. Conformity with the National ITS Architecture is another area to address in the up-front planning of a project.

Related to the concept of up-front planning is the need to address problems as they occur, early rather than later. There is considerable cost impact if problems are swept under the rug or deferred. But surfacing problems early requires open communications; addressing them requires team building.

System themes

Don’t Build If You Can Buy. Whenever possible, buy existing software products instead of building custom software from scratch. Product offerings, ranging from system components that must be integrated to commodity products, are available for many types of ITS systems. Building new, custom software adds risk to a system. It also adds cost, since for off-the-shelf products, the development costs are amortized across many customers. To allow for an off-the-shelf solution, you must have the flexibility (see the flexibility management theme above) to relax requirements and buy a system that provides the “best fit.” This requires being able to live with the recognition that an off-the-shelf solution probably won't match your vision of the ideal solution. An off-the-shelf product may not do everything on your wish list, but an 80% solution may be good enough. "Do not seek exactness when only an approximation of the truth is possible." [Aristotle]

The off-the-shelf theme is one that the private sector particularly endorses. Now we recognize that buying off-the-shelf software is not a silver bullet. (See *Build/Buy Decisions*, Chapter 10.) The theme here is not that you should always go with an off-the-shelf solution. But rather, that you should give it serious consideration, and use it *where it makes sense*. If you have requirements that preclude off-the-shelf solutions, think them through and determine how important they are to have and at what cost. If you then decide that you must have the software built to meet truly unique situations, and building it is within the state-of-the-art, by all means do so. But give the buy option a fair chance. If you later decide to upgrade, the need for customization will be based upon operational experience, not conjecture. Of course, any such customization is done as a teaming process with your supplier to determine what is readily doable and what isn't. Then carry out the customization a piece at a time-which brings us to our next theme.

While private-sector vendors are the most outspoken on this topic, they are not the only ones to endorse off-the-shelf software solutions. Consider the experiences of one public-sector project manager who had been through a difficult acquisition to build ITS software. He told us that although he had not considered the off-the-shelf option, in retrospect he should have. It seems likely that what had been built could have been bought off-the-shelf with much less angst. Even the Department of Defense, which has a long history in building software, has taken steps in recent years to encourage the use of off-the-shelf solutions.

Take Bite-Size Pieces. A frequent problem with software acquisitions is that they attempt to do too much. To be sure, it's desirable to have an ambitious long-term vision, to "think big." But this is best accomplished a step at a time. At all points, resist requirements or scope creep.

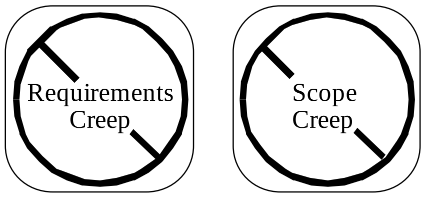
Consider establishing a baseline capability, buying an off-the-shelf product if possible, as the first bite-size piece. Develop an overall framework and build in the flexibility for future expansion. Once a high-quality baseline is fielded, it can serve as the foundation for further acquisition activities, allowing more functions to be added later.

There are a number of advantages in taking bite-size pieces:

- It makes the acquisition more manageable. Large projects may seem technically doable. But when they fail, it's not because of the technology, but because they go beyond the state-of-the-art (or at least the state-of-the-practice) of software management.
- As a system becomes larger, error rates increase, and the mean time to failure decreases. [Putnam and Myers, 1992, page 136] There is some

evidence that as added requirements interact in unanticipated ways, system complexity goes up and reliability goes down in proportion to the *square* of the size. So if you cut back on the size, you will get disproportionate benefits.

- "Because the effort to build software increases disproportionately faster than the size of the software, a reduction in size will increase development speed disproportionately." A rough guideline is that cutting the size in half reduces the overall effort by two-thirds. [McConnell, 1996]
- With smaller pieces, something gets fielded sooner. You will find that as new systems are implemented, they usually get used in unexpected ways. By allowing the users to "kick the tires" of a tangible product, later pieces can then benefit from this operational experience, addressing unanticipated needs of the users. This is preferable to acquiring a system that addresses all the needs conjectured at the outset, only to find that much of the effort was wasted on developing features that never got used. [Costantino *et al.*, 1995] Moreover, having a tangible product to show is an effective way to build political support for achieving your long range objectives. It also improves morale as developers and managers see the fruits of their labor.
- With a smaller acquisition and its shorter schedule, many of the same team members will still be around at the conclusion of the project. A major problem that plagues big acquisitions is high turnover in personnel, with few of the original players remaining at the end. This is true in both the public and private sector. One sector's turnover negatively impacts the other sector because of the loss of corporate memory. The new team members may not accept the original requirements, and even though the system is "done" (per the original requirements) additional work is needed to satisfy the perceived needs of the new players. With a shorter acquisition, the problem is reduced.
- Long acquisitions perpetually try to chase the ever accelerating pace of advances in computer technology. By the time a new advance is accommodated, yet another one appears.



How do you go about resisting requirements or scope creep, keeping the pieces small, without unduly limiting your vision of the ultimate goals and objectives? One way is to constrain the functionality so it can reasonably be expected to be implemented in a relatively short schedule. (However, be careful about imposing an unrealistic short schedule. The schedule should be based on what you are trying to achieve, not on wishful thinking.) If you find that a piece will take longer than a year to implement, subdivide it into smaller pieces. "Big steps are killers." Studies of software organizations with successful track records show that they take bite-size pieces needing no more than nine months from requirements to delivery. [Jones, 1997]

Related to the *design to schedule* approach discussed in the preceding paragraph is *design to cost*. Develop a list of mandatory and optional features, and only tackle those that can be implemented within cost constraints.

Another approach is to have a task-order contract, with only the first one or several tasks "turned on." Each task can include an activity to plan the next one. Additional tasks are turned on one at a time. In effect, a large acquisition is turned into a series of smaller ones. For this to work, the contract must have the flexibility to allow you to chart your course as you go.

There have been a number of notable ITS successes achieved through the use of incremental development. In Houston, Texas, a basic traffic management center was established and expanded to incorporate other agencies over time as they saw the benefits of closer coordination. This was seen as more effective than attempting to have all the parties reach up-front agreements on a coordination plan. In Montgomery County, Maryland, basic traffic data on cable TV led to public support for more capabilities. It would have been much more difficult to generate public support for a comprehensive paper concept when they had no prior experience with what the data could offer. [Federal Highway Administration, 1996.]

Counterparts to themes: what not to do

Several philosophies are commonly heard on software acquisitions that run counter to the themes. They almost invariably lead to trouble:

- "holding the contractor's feet to the fire"
- squeezing the schedule ("we know it's unrealistic, but...")
- building everything in one fell swoop and turning it all on at once (the "big bang theory")
- asking for more than what's in the contract (the "free lunch")
- picking the right contractor and leaving them alone ("laissez-faire")
- being overly optimistic with unrealistic expectations ("sure it's risky, but if all the chips fall in place...")
- by the contractor: low-balling the project to "get in the door" and position oneself for future work, counting on engineering changes to "get well"
- on a fixed-price contract, requiring the contractor to supply cost details that are suitable for cost-reimbursement contracts instead of paying for completed tasks or milestones

Applying themes to software acquisition activities

In Part Three and Part Four of this document (see Volume II), we relate the themes to the various activities that occur on a software acquisition. Many of the themes will recur again and again. When we come across a theme, we'll call it out. Indeed, many of our recommended activities are just specific illustrations of one of the themes, a variation on the "software acquisition melody."

Before closing this chapter, we provide a quote that nicely summarizes several of our themes:

> *"A most important ingredient for the success of a software project is an environment of communications and mutual trust between the buyer and seller. Many buyers believe that sellers are knowledgeable and experienced in development and that they should be left to their own devices. Too often the buyer relies totally on the contract as a management vehicle. The buyer's approach is to consider problems that arise to be the seller's problems and to expect contractual pressure on the seller to solve them. The adage, "Hold the contractor's feet to the fire," has been a common management mistake. When a problem occurs, the seller is therefore always to blame. This intractable approach has proved unworkable time and again.*
>
> *"The contract represents the best written communication for the requirements of the effort at the initiation of development. It is only a document, however, and cannot take the place of effective management by the buyer and seller. The objective of the development is not the exercise of the contract—it is the delivery of a viable capability. To achieve a successful delivery requires communication and cooperation by both the seller and buyer—in other words, it calls for a team effort." [Marciniak and Reifer, 1990]*

- *Build your software acquisition around certain themes that should recur throughout the various acquisition activities:*
  - *– People themes, which are akin to partnering.*
  - *– Management themes, on how to approach the acquisition.*
  - *– System themes, relating to the end product.*
- *The themes can guide you as to the best practices to employ in approaching your software acquisition.*
- *Collectively, the themes address the problems commonly associated with software and represent our response to the overarching theme that "software is different."*

CONCLUDING REMARKS TO VOLUME I

This concludes Volume I of *The Road to Successful ITS Software Acquisition*. In this volume, we explained how software is different and introduced a set of themes for dealing with those differences. Volume II builds upon the themes. It discusses the activities associated with a software acquisition and shows how the themes can be used to guide the activities.

We invite you now to turn to Volume II.

REFERENCES

E. Bersoff, V. Henderson, and S. Siegel, *Software Configuration Management: An Investment in Product Integrity*, Prentice-Hall, 1980

Booz-Allen, *FHWA Federal-Aid ITS Procurement Regulations and Contracting Options*, August 1997

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. Cappelletti and P. Gerdes, *Nondevelopmental Item (NDI) and the System Acquisition Process*, MITRE Corporation Technical Report MTR 94W22, 1994

D. Carney and P. Oberndorf, "The Commandments of COTS: Still in Search of the Promised Land," *Crosstalk: The Journal of Defense Software Engineering*, Vol 10 No 5, May 1997

M. Christel and K. Kang, *Issues in Requirements Elicitation*, Software Engineering Institute Technical Report CMU/SEI-92-TR-12, 1992

The Condensed Guide to Software Acquisition Best Practices, 1997. pamphlet available from Software Program Managers Network

J. Costantino *et al.*, "Air Traffic Control: Lessons Learned for Surface Transportation," *ITS Quarterly*, Vol III No. 1, 1995.

Department of Defense, *Guide to Integrated Product and Process Development (IPPD)*, Version 1.0, February 5, 1996. Available at over the Internet in HTML and Word formats at <URL:http://www.acq.osd.mil/te/survey/survmain.html>

M. Evans and J. Marciniak, *Software Quality Assurance and Management*, John Wiley & Sons, 1987

D. Farbman, "Myths That Miss," *Datamation*, pp. 109-112, November 1980

Federal Highway Administration, *Key Findings from the Intelligent Transportation Systems (ITS) Program: What Have We Learned?*, U.S. DOT Publication

FHWA-JPO-96-0036, 1996

J. Ferguson *et al.*, *Software Acquisition Capability Maturity Model (SA-CMM^{SM}) Version 1.01*, SEI Technical Report CMU/SEI-96-TR-020, 1996

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

D. Garlan and D. Perry, "Introduction to the Special Issue on Software Architecture," *IEEE Transactions on Software Engineering*, Vol 21 No 4, 1995

W. Gibbs, "Software's Chronic Crisis," *Scientific American*, pp. 86-, September 1994

R. Glass, *Modern Programming Practices*, Prentice-Hall, 1982

R. Higuera and Y. Haimes, *Software Risk Management*, Software Engineering Institute Technical Report 96-TR-012, 1996

R. Higuera *et al.*, *Team Risk Management: A New Model for Customer-Supplier Relationships*, SEI Special Report CMU/SEI-94-SR-5, 1994

B. Horowitz, *The Importance of Architecture in DOD Software*, The MITRE Corporation, M91-35, July 1991

W. Humphrey, *Introduction to Software Process Improvement*, SEI Technical Report CMU/SEI-92-TR-7, revised June 1993

W. Humphrey, *Managing the Software Process*, Addison-Wesley, 1989

IEEE, *IEEE Recommended Practice for Software Acquisition*, IEEE Std 1062-1993, 1993

IEEE, *IEEE Recommended Practice for Software Requirements Specifications*, IEEE Std 830-1993, 1993a

IEEE, *IEEE Standard for Software Maintenance*, IEEE Std 1219-1993, 1993b

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

J. Marciniak and D. Reifer, *Software Acquisition Management: Managing the Acquisition of Custom Software Systems*, John Wiley & Sons, 1990

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

MITRE Corporation, *Software Reporting Metrics*, MTR-9650 Rev 2, November 1985

M. Paulk, *Key Practices of the Capability Maturity Model, Version 1.1*, SEI Technical Report CMU/SEI-93-TR-025, February 1993

V. Pearce, "Procurement: Hard Work Pays Off," *Traffic Technology International*, pp. 70-, Oct/Nov 1997

T. Pigoski, *Practical Software Maintenance: Best Practices for Managing Your Software Investment,* John Wiley & Sons, 1997

P. Place, P. and K. Kang, K., *Safety-Critical Software: Status Report and Annotated Bibliography,* SEI Technical Report CMU/SEI-93-TR-005

B. Prasad, *Concurrent Engineering Fundamentals (Volume 1: Integrated Product and Process Organization; Volume 2: Integrated Product Development),* Prentice-Hall, 1996

The Program Manager's Guide to Software Acquisition Best Practices, 1997. available from Software Program Managers Network

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

T. Royer, *Software Testing Management: Life on the Critical Path*, P T R Prentice Hall, 1993

T. Saunders, B. Horowitz, and M. Mleziva, *A New Process for Acquiring Software Architecture*, The MITRE Corporation, M92-B126, 1992

The Standish Group, *Charting Seas of Information Technology,* 1994

STSC (Software Technology Support Center*), Software Configuration Management Technology Report*, <URL: http://stscols.hill.af.mil/cm/REPORT.HTML>, 1994