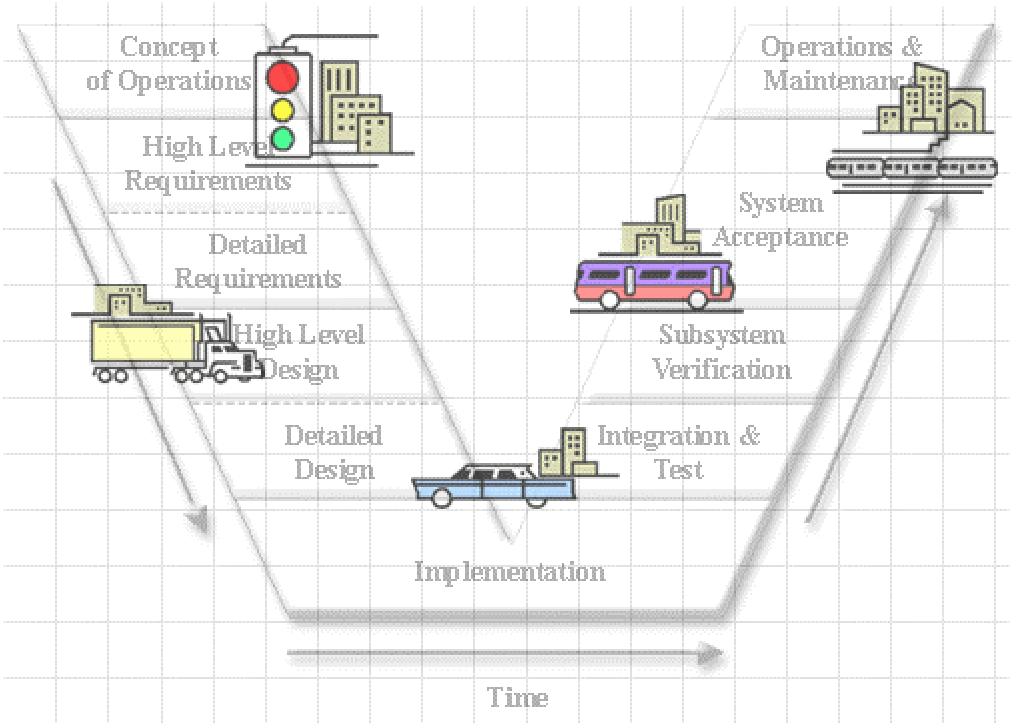
Building Quality Intelligent Transportation Systems Through Systems Engineering

April 2002



Prepared for

Intelligent Transportation Systems
Joint Program Office
US Department of Transportation

By Mitretek Systems, Inc.

Technical Report Documentation Page

1. Report No. FHWA-OP-02-046	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Building Quality Intelligent Transportation Systems Through Systems Engineering		5. Report Date April 2002
		6. Performing Organization Code
7. Author(s) Paul J. Gonzalez		8. Performing Organization Report No.
9. Performing Organization Name and Address Mitretek Systems, Inc. 600 Maryland Avenue, SW, Suite 755 Washington, DC 20024		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. DTFH61-00-C-00001
12. Sponsoring Agency Name and Address Department of Transportation Intelligent Transportation Systems Joint Program Office 400 Seventh Street, SW – Room 3416 Washington, DC 20590		13. Type of Report and Period Covered
		14. Sponsoring Agency Code HOIT
15. Supplementary Notes William S. Jones – Task Manager		

16. Abstract

This monograph is intended to introduce the topic of systems engineering to managers and staff working on transportation systems projects, with particular emphasis on Intelligent Transportation Systems (ITS) projects. Systems engineering is a discipline that has been used for over 50 years and has its roots in the building of large, complex systems for the Department of Defense. Systems engineering is an approach to building systems that enhances the quality of the end result and the expectation is that its application to transportation systems projects will make those projects more effective in developing and implementing the systems they are intended to build. Although applying systems engineering techniques on a project doesn't guarantee success, not following a systems engineering approach is a strong recipe for failure.

This monograph presents a common approach to systems engineering, one that is followed in many industries and domains, not just by Defense Department contractors. This approach emphasizes the combination of technical and management activities that produce a disciplined approach to building systems. And, although anyone from a technical or scientific discipline can learn to practice good systems engineering techniques, the most effective systems engineers are those who also bring domain knowledge about the system being built. Since the domain knowledge in transportation systems involves knowledge of transportation systems, it is important to being familiarizing transportation engineers about this discipline.

This monograph is intended for use in conjunction with the systems engineering courses being offered by the National Highway Institute.

17. Key Word Systems Engineering, Intelligent Transportation Systems, ITS, Transportation System Projects	18. Distribution Statement No Restrictions This document is available to the public.		
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 81	22. Price NA

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

Table of Contents

List of Tables and Figures

Executive Summary

Purpose and Intended Audience

Systems engineering is an approach to building systems that enhances the quality of the end result. It has its roots in the development of large, complex systems for the Department of Defense. Systems engineering has been around for over 50 years and there are many books and articles that cover it in great detail. Our goal is to introduce systems engineering to managers and staff working on Intelligent Transportation Systems (ITS) projects, if they aren't already familiar with its practice. We believe that they will become more effective in acquiring, developing, and implementing ITS systems by following systems engineering practices. We prepared this monograph to introduce transportation professionals to established systems engineering practices that have proven successful in other domains.

Our intended audience includes:

- ITS project managers
- ITS project staff
- Contractors and their staff working on ITS projects
- Anyone else interested in systems engineering, particularly on software-intensive systems

Although our primary focus is on ITS project managers, we hope our coverage of this topic encourages others to learn more about this important area. Applying systems engineering techniques on a project doesn't guarantee success; not following a systems engineering approach, however, is a strong recipe for failure.

What is Systems Engineering?

If you ask a group of systems engineers to define "systems engineering," you might get more definitions than there are members of that group. In fact, when the International Council on Systems Engineering (INCOSE) was formed, the "entrance fee" to the first meeting was a definition of "systems engineering" from anyone who wanted to attend. No two definitions were exactly alike, but they tended to fall into one of four categories:

- Those who considered it a function solely of "systems engineers," and consisting only of technical activities

- Those who saw it as a function solely of systems engineers, but consisting of both technical and management activities

- Those who believed that it consisted only of technical activities that anyone from a technical or scientific discipline could do

- Those who saw it as set of technical and management activities that anyone from a technical or scientific discipline could do

Our position on the subject falls into the last category. As we discuss throughout this paper, systems engineering combines technical activities and management activities to produce a disciplined approach to building systems.

Although anyone from a technical or scientific discipline can learn to practice good systems engineering techniques, to be most effective as a systems engineer, a person must have *domain knowledge* about the system being built. *Domain knowledge* is a fundamental understanding of the technology and functions involved in the system being built. In an ITS system, for example, *domain knowledge* includes transportation engineering or transit system management. Without *domain knowledge*, a systems engineer is not as effective. However, if you have to choose between using systems engineers who don't have *domain knowledge* and not using any systems engineers, use the systems engineers and complement them with staff who do have *domain knowledge*.

Why Does Systems Engineering Matter to ITS Projects and Project Managers?

What every ITS project manager wants is a successful system at the end of the project, with "success" measured by how well the system satisfies the requirements of the people who use it. So a goal-oriented ITS project manager wants to use any tools or techniques that help achieve success. Systems engineering provides those tools and techniques.

Systems engineering helps accomplish four key activities that impact a project's success. These are:

- Identify and evaluate alternatives
- Manage uncertainty and risk in our systems
- Design quality into our systems
- Handle program management issues that arise

Identifying and evaluating alternatives is important as we attempt to determine which alternative system design and implementation offers us the best chance to succeed. We needed to measure the feasibility of each alternative from three different points of view: technical feasibility, cost feasibility, and schedule feasibility. *Technical feasibility* addresses whether we can build, maintain, and operate a system alternative, given the technology and people available to us. *Cost feasibility* looks at whether we can build, maintain, and operate a system alternative with the funds available for it. *Schedule feasibility* considers whether we can build a system alternative within the time frame allotted for its development. Usually we have to make trade-offs; one alternative may cost less than another, but we may be able to build a second alternative faster than the first. We have to decide which offers the better value. If several alternatives fit within the technical, cost, and schedule parameters that we've set, it may come down to which alternative offers the least implementation risk.

Managing risk and uncertainty in our systems development efforts is important because we want to avoid mistakes or potential problems that threaten the success of our work. Systems engineering focuses on three aspects of risk management: identification, analysis, and mitigation.

Designing quality into our systems is done by addressing those factors that can negatively affect quality. Paraphrasing the International Organization for Standardization (ISO), we can define quality as "the totality of features of a system that bear on its ability to satisfy stated or implied needs." Among the factors that can negatively affect the quality of a system are its *complexity*, its *inflexibility*, its *lack of standardized components*, and its *reliability and availability*. A *complex* system is hard to use and maintain. While it may be necessary for a system to be complex, our goal should be to keep it as simple as possible. An *inflexible* system doesn't adapt well to change; when its environment changes or when we must add to it or modify it to deal with changes in our needs, it may fail us. Systems that *lack standardized components* are difficult to maintain. When we must replace some part of the system, we may not be able to find a replacement part. This can increase overall system maintenance costs or cause the system to fail while operating. We can't use systems effectively if they are not *reliable* and *available*. An unreliable system breaks down while we're trying to use it; an unavailable system isn't there to be used when we need it.

Handling project management issues that arise is easier to do if we have a good project plan to start with. A good project plan should be complete, comprehensive, and communicated. We can ensure the completeness and comprehensiveness of the plan by:

- Including all tasks that we must perform
- Accurately estimating the resources required to accomplish each task
- Assigning the appropriate resources to each task
- Defining all dependencies among tasks
- Identifying all products or other criteria whose completion signifies that a task is done
- Determining how to measure progress against plan when managing our project

We must also communicate the plan to everyone that it affects. We must tell those people assigned to work on project tasks:

- What work they are responsible for
- When they should begin a task
- When they should complete a task and how they will know when a task is done
- Who else will be working on that task
- What tasks are dependent on the one they are working on and what tasks their task depends on
- What non-people resources they will need to do their job

Having done that, we must then track each task, measure its progress, revise the overall plan if needed, and identify and address any obstacles that impede our progress. These are standard project management activities, but project management is an important element in a good systems engineering program.

Systems Engineering Impacts on System Implementation

Systems engineering is a process, not just a set of tools. As such, systems engineering activities occur throughout the system development life cycle. A common set of stages in a system development life cycle and the systems engineering technical activities that accompany them include the following:

- **Conception**. The stage in which the need for a system (or major system enhancement) is first identified. The principal systems engineering activities in this stage revolve around feasibility assessment. Is the system feasible? Can a feasible system be built in a reasonable time and at a reasonable cost? How much time will we need to build this system? How much should it cost? These are the types of questions that a systems engineer focuses on during this stage.

- **Requirements Analysis**. During the requirements analysis stage, the systems engineer focuses on ensuring that the requirements defined for the system state <u>what</u> needs to be done rather than <u>how</u> it should be done. The systems engineer also works at ensuring that the requirements defined are clear, complete, and correct. The systems engineer schedules reviews with the stakeholders in the system to ensure that all parties involved in building the system have the same understanding of what each requirement means. Stakeholders include the contractor selected to build the system.

- **Design**. During this stage, the systems engineer helps flesh out the details of the system, helping make decisions about the best way to satisfy the system's requirements. To help overcome technical uncertainty, the systems engineer may conduct trade-off studies, use modeling and simulation to analyze potential system performance, build prototypes to assess the technical feasibility of a proposed solution, or perform in-depth analyses of different technologies to assess their applicability to the system under development. The systems engineer also conducts design reviews with project stakeholders, to ensure that the design approach selected is consistent with their needs and expectations.

- **Implementation**. Systems engineering activities during this stage (sometimes called ***Development***) focus on ensuring that what gets built matches the agreed-upon design. One specific subset of systems engineering activities during this stage is software engineering, designed to ensure the quality of the software in the system. These activities include walk-throughs of developed programs (where programmers review the work of another programmer, to determine whether any errors exist). In addition, software engineers define standards for code development and ensure that these standards are followed. But the systems engineering activities aren't solely directed at software. It is also important to ensure that hardware developed (or modified) during this stage also matches the agreed-upon design. Hardware inspections are one technique for quality control during this stage.

- **Testing**. Systems engineers create complete and comprehensive test sets that effectively exercise the system and its components. They also analyze test results and assess the degree to which the system satisfies its requirements.

  Testing is important to software development, because it uncovers errors in the logic of the programs. Although it is unlikely that all execution paths through a program can be covered, systems engineers focus on identifying test coverage to ensure that mission-critical functions are thoroughly tested.

  Hardware components need to be exercised as part of the testing process to ensure that they perform as expected and as the system requires them to. And you should recognize that hardware testing needs to take place under realistic conditions of use, not just a "laboratory" environment. Some hardware components, e.g., global positioning system receivers, may work quite well under controlled climatic conditions, yet fail when exposed to realistic weather conditions, such as below-freezing temperatures.

- **System Acceptance**. Some of the key systems engineering activities during this stage look at the training of system users and the approach for fielding the system. New systems, or enhanced versions of existing systems, need to be fielded in a manner that does not disrupt normal business operations. Systems engineers plan these "rollout" activities to effect a smooth transition to the new system. This stage also involves the final testing of the system by its users, to ensure that it meets their requirements.

- **Operation and Maintenance**. After the system is fielded, systems engineers focus on ensuring that the system continues to meet its performance goals. Should problems arise, they will diagnose the causes and determine how to solve the problem with minimal delay. In addition to fixing problems that occur, systems engineers also assess potential enhancements and upgrades to system capabilities. This is particularly critical, considering the normal incremental, evolutionary approach to fielding modern transportation systems.

In addition to the technical activities that systems engineers perform, systems engineering also plays an important role in project and program management. Systems engineering activities in project and program management span multiple system development life cycle stages. Management activities associated with systems engineering fall into four major categories:

- **Project planning and control**. Planning is an essential element of systems engineering. Without good plans, it's difficult to know what to do and when to do it. But control is also important. Control involves two key areas: 1) monitoring project activities to determine task status, and 2) initiating corrective action when problems arise. To monitor project tasks, the systems engineer sets up feedback mechanisms that provide accurate and timely status information. These feedback mechanisms also identify problems that impede progress. When a problem arises, the systems engineer shifts into analysis, to assess the impact of the problem and to determine ways to eliminate it or to minimize its effect.

- **Technical planning**. Technical planning primarily involves determining how your systems engineering organization is structured and what activities it will perform. For each project, the systems engineer creates the Systems Engineering Master Plan (SEMP), which describes how systems engineering is organized on the project and how systems engineering activities are carried out. You must decide how you will involve your contractors, if at all, in conducting systems engineering activities during your project.

- **Technical audits and reviews**. At each project milestone, you conduct a technical audit and review to determine how complete the technical work on the project is, before continuing. The SEMP spells out which technical audits and reviews you will conduct, when you will conduct them, who will conduct them, and what effect the outcome of the technical review and/or audit will have on the project's next steps. All of these reviews and audits should be formal and documented as part of the project's audit trail.

- **Cost estimating**. Estimating project costs is a management activity. The systems engineer takes a global view of the project and considers all costs, the "life cycle" costs of the project. Thus, the systems engineer will estimate not just development costs, but also operation and maintenance costs (e.g., repair costs, replacement costs – including upgrades), the costs involved in fielding a system implementation (e.g., supply costs), and training costs.

Introducing Systems Engineering Practices on ITS Projects

Every ITS project has "success" as a goal. Success is delivering a sound, reliable system that meets user needs and gets done within schedule and within budget. Your best chance at achieving success comes from introducing good systems engineering practices on your project from the start and continuing to use them until the project ends (and beyond). Let's consider what some of these practices are.

Planning

Good systems engineering starts with planning. Your plan is your roadmap for getting to success. Systems engineering looks at the plan from the broadest possible view, considering all aspects of the project – from onset to operation and maintenance.

Systems engineering plans each step along the way and plans how you're going to address the potential road blocks that you may find as you move from beginning to final delivery. While it isn't always possible to plan long projects in detail at the beginning, you can usually plan the early stages of a project in detail and the rest at a higher level. Then, as you accomplish the tasks planned in detail, you learn enough about how the rest of the work should be done and can increase the level of detail in subsequent months. This is a continual planning approach that lets you enhance your chances of planning the right work and planning the work right.

Your plans should synchronize with your cost estimates for the work to be done. As you refine your plans, you must refine your cost estimates. Your initial cost estimate for your project sets a ceiling that likely is difficult to raise. Your re-planning and re-costing efforts must aim at staying within that cost ceiling. However, if it appears that the original target was underestimated, your continual re-planning and re-costing gives you an early indication of cost growth while you can still control the growth.

Reviews and Audits

In a manufacturing process, quality is maintained through inspection of the finished goods and of the goods in process. If quality inspectors detect flaws early enough in the manufacturing process, the manufacturer can correct the cause of the flaws and might salvage the product. If the quality inspectors don't detect the flaws until the product is finished, the flawed product is kept from the market place, protecting the manufacturer's quality reputation.

In systems development, the quality inspection process consists of the reviews and audits that you perform. You review documents, requirements, designs, program code, test results, and anything else that might give you an indication of the quality of the system that you're building. Audits ensure that you know, at any time, what your system consists of and where all of its parts reside. Many texts on systems engineering, and even the systems engineering standards that exist, describe the types of reviews and audits that you can conduct. As you introduce the practice of systems engineering, these should be your references.

Uncertainty and Risk Management

We can never be certain about the future. However, we must make decisions when implementing a system that assume some confidence in what the future holds. When uncertainty is cause by lack of knowledge or lack of information, we can apply systems engineering practices to help reduce that lack. Systems engineers use mathematical and software tools and engineering techniques to increase our knowledge and information. Prototyping, for example, is frequently used to gain a better understanding of what human-computer interfaces best suit the needs of our system's users. We can also use it as a "proof of concept" technique, where we integrate components (hardware and software) that have not been integrated before. In integrating these components, we learn what works and what doesn't work as an integrated unit. We also learn how to integrate the components effectively (or at least, how not to).

Another tool that systems engineers use is simulation modeling. A simulation can test different assumptions about how a system will work, under different conditions of load, before that system is built. That helps identify the bottlenecks in the system that negatively affect performance and provides us clues about how to eliminate those bottlenecks. Simulations can also help us identify when and where processes interfere with one another. This allows us to revise the processes and improve workflow.

Any tool or technique that reduces our uncertainty helps reduce the risk involved in building a system. The more we know about the possible outcomes of a decision or approach we plan to take, the more easily we can avoid the undesirable outcomes. This is one aspect of *risk management*. But there are other aspects as well. Risks can come from "threats," i.e., outside events that can have a negative impact on our efforts. "Threat analysis" identifies those potential events that may harm us and allows us to estimate the potential cost to our project of each event's occurrence. Once we have identified "threats," we can plan our strategy for mitigating their ability to harm us.

A Final Word

Systems engineering is hard work. It brings as a reward the increased probability that we will successfully implement a useful, reliable system. This paper won't make you a systems engineer, but it does introduce the topic and provides you with a framework for understanding its value. We hope you find it useful.

1.0 Introduction

Purpose of This Document

This document introduces the concept and practice of systems engineering, and discusses why it applies to the acquisition, development, and fielding of Intelligent Transportation Systems (ITS). Systems engineering is a vast topic area and you can find many books and articles that cover it in detail. You'll find some of them listed in the References at the end of this document. Reading this document (or those books and articles) won't turn you into a systems engineer; that takes experience as well as more in-depth knowledge of systems engineering topics. But our goal is to provide you with an overview of the subject and, we hope, spark your interest in pursuing it in more detail.

Systems engineering has proved beneficial in building systems in a number of different areas. We believe that its use in implementing ITS systems will lead to better systems and more successful projects. By the end of this document, we hope you agree.

Some Conventions Used in the Document

Throughout this document, we'll mark "key" ideas by using a key symbol and some text that summarizes the idea we want to emphasize. This marker looks like the following example:

This is a "key" idea example

We'll float the key idea box in the margin next to the text that we are emphasizing. This should help you extract the important concepts you'll use when implementing systems engineering in your projects.

We'll also use some examples to get our points across. We'll use examples that are related to transportation and ITS projects whenever possible, but may bring in ones from other areas when they help make the point. We'll mark our examples with simple graphics as well. When we want to talk about schedules, costs, and technical areas, we'll use the following graphics:

Schedule

Cost

Technical

Intended Audience

The audience for this document is any or all of the following:

- Project managers of ITS projects
- Project team members on ITS projects
- Contractors and staff on ITS projects
- Anyone else interested in quality results in systems implementation

It is primarily geared, however, to ITS project managers, to help them succeed in their efforts. Applying systems engineering in a project effort doesn't guarantee success; external factors can always cause a project to fail. However, experience shows that not applying a systems engineering approach when implementing a system is a good way to fail. So, let's look at what systems engineering is all about.

What is Systems Engineering?

A web page entitled "Systems Engineering Definitions" at a major Virginia university[1] has the following definitions (unchanged from how they appear on that page) from the university's systems engineering faculty:

- Systems Engineering is the design and analysis of large-scale systems, where these activities add value to the enterprise or operation.

- Systems Engineering is a philosophy, grounded on the arts and sciences, supported by modeling and optimization methodologies, for the purpose of improving our understanding of the uncertain nature of the system and improving the decisionmaking process.

- Systems Engineering is the application of scientific principles to the design, development, implementation, and control of complex large-scale compositions of resources and processes which interact to achieve a common objective.

- Systems Engineering is just good problem solving.

- Systems Engineering is a structured approach to solving problems--a creative, iterative, decision-making process by which our understanding of logic, mathematics, and science is joined with our understanding of human and institutional needs, wants and limitations to conceive and refine alternatives that serve specific purposes within specific problem contexts.

- "A system is a set of elements so integrated as to aid in driving toward a defined goal." "Systems Engineering = Systems Analysis," and "Systems Analysis equals operations research plus policy analysis."

[1] Taken from a University of Virginia Department of Systems Engineering web page, http://freney.sys.virginia.edu/sys-def.html.

Our intent in providing such varying definitions is not to confuse you (or to poke fun at academia), but to illustrate that there is no one accepted, "perfect" definition of the term.

Why is Systems Engineering So Difficult to Define?

The term, "systems engineering," was coined in the 1950s to describe a process used by engineers working on large and complex systems, usually incorporating computers and software. A significant body of work in systems engineering derives from the US military's efforts to build large command, control, and communications defense systems and the need to ensure that those systems worked when fielded. Since the Department of Defense (DoD) hired contractors to build most of those systems, the DoD wanted those contractors to follow an approach that gave the highest possible likelihood of overall success, regardless of whom the DoD selected to build a particular system. The DoD developed systems engineering standards and required contractors to conform to them. Over time, these standards have evolved and matured and become accepted by more than just defense contractors.

But systems engineering applies to the building of any large system, regardless of type. As practitioners began to write textbooks for others to use, some followed the DoD model and others looked at the basic principles involved, without necessarily using the formalism of the DoD model. Thus, there are diverse descriptions and definitions throughout the literature of systems engineering. If you look at any text among the references in this paper, you will find several with at least two different definitions.

People working in this field don't always agree on what constitutes systems engineering. When the International Council on Systems Engineering (INCOSE) was formed in 1990, all who attended the founding session had to contribute a definition of systems engineering. No two were the same, but they could all be grouped into one of the following categories:

- Those who considered it a function solely of "systems engineers," and consisting only of technical activities

- Those who saw it as a function solely of systems engineers, but consisting of both technical and management activities

- Those who believed that it consisted only of technical activities that anyone from a technical or scientific discipline could do

- Those who saw it as set of technical and management activities that anyone from a technical or scientific discipline could do

The technical activities in systems engineering involve an understanding of such tools as queuing theory, modeling and simulation, and linear programming. It also helps to have a sound grounding in probability and statistics. But, to be most effective, you also need *domain knowledge*. *Domain knowledge* is a fundamental understanding of the technology and functions involved in the system under discussion. For example, in an ITS system, *domain knowledge* may involve transportation engineering. In an air traffic control system, *domain knowledge* includes understanding what an air traffic controller does.

Why Use Systems Engineering?

A lot of what's involved in systems engineering is common sense and forethought – and good engineering principles that you learn in any engineering program. Systems engineering is practiced along with many other engineering disciplines, whenever complex systems are involved.

One area where systems engineering is particularly important is in building software-intensive system, i.e., systems where software is a critical component for overall system success. Software-intensive systems are notorious for being difficult to implement successfully. The Standish Group, a national research firm, found, in a study they conducted in 1994, that only 16% of all software-intensive system development projects were "successful. [2]" By "successful," they meant that the project was completed on time, within budget, and with all of the initially planned functionality delivered. In the same study, they found that 31% of all such projects were "failures," i.e., were canceled prior to completion. This is a failure to success ratio of nearly 2 to 1. The remaining projects were completed with one or more of the following conditions: behind schedule, over budget, or with less functionality in the systems than originally planned. Four years later, the Standish Group repeated its study and found improvement. The percentage of "successful" projects had gone up to 28% and the number of "failures" had dropped to 28%. The failure to success ratio was now 1 to 1. Figure 1 below illustrates the Standish Group results from their two studies.

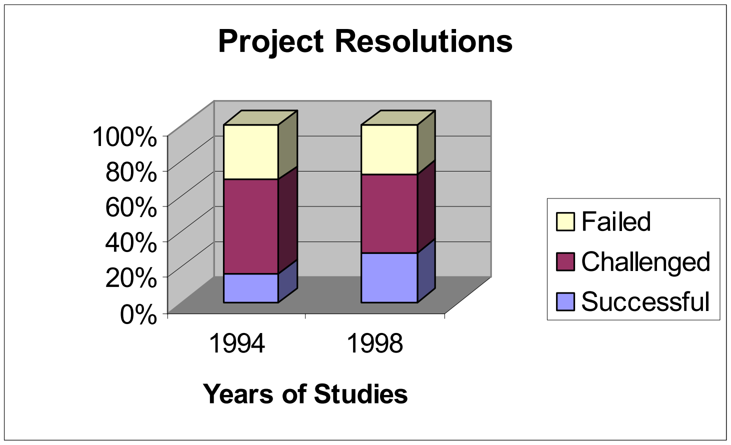


Figure 1
Project Resolutions

[2] "Turning Chaos into Success," *Software Magazine*, December 1999, pg. 30.

The use of systems engineering techniques wasn't the sole cause for the improvement, but their use did play a key role.

The Challenges in System Engineering

When we apply systems engineering on any project, there are at least four major things that we want to accomplish. These are:

- Identify and evaluate system alternatives
- Manage uncertainty and risk in our systems
- Design quality into our systems
- Handle any program management issues that arise

Identifying and Evaluating System Alternatives

There is no one right way to build a system.

When building a system, we make choices about how to accomplish our goal. Systems engineering helps us make better choices by providing a framework within which to make those choices, one that helps us analyze and understand the consequences of our choices. If we can compare the consequences of our choices on an even par, we improve the likelihood that we will make a good choice. Again, we're not dealing with a crystal ball that predicts the future accurately. We're going to use analysis and judgment and make estimates about what will happen.

When we build a system, we use systems engineering to do several things. First, systems engineering techniques help us define a set of alternative ways of building it, that is, to define a set of "system design trade-offs." Second, system engineering provides techniques for evaluating each system design trade-off and determining which one gives us the best chance of successfully building the system we want. We do this by assessing three things:

- ***Technical feasibility*****:** This is our judgment on whether the system we want can and should be built using the technology in our design alternative. Let's take the following example: We want to be able to process the input from 300 different sensors and have the information available at our Transportation Management Center (TMC). One design alternative is to use existing communications lines to transmit the feed from all 300 sensors. A second alternative is to use upgraded communications lines, with increased capacity, to transmit the feed from all 300 sensors. A third alternative is to use a combination of existing communications lines and new lines, with part of the sensor array on each set of lines.

  We conduct a technical trade-off analysis to assess each alternative and compare it to the others. First we determine if the alternative is feasible. If, for example, the existing communications lines are incapable of handling all 300 sensors, we can reject that alternative immediately. That alternative can't solve our problem. If all three alternatives are feasible, then we consider other factors to determine which gives us the best solution. From a technical viewpoint, the best alternative is the one that gives us the most capacity, because that provides the greatest flexibility and potential to handle growth, is the most reliable, because we avoid transmission errors and interruptions, and the most available (which may require building in some redundancy), since that provides the most consistent service. However, the best technical solution isn't necessarily the best <u>solution</u>. To determine which one is best overall, we also need to consider cost and schedule factors.

- ***Cost feasibility*****:** This is our judgment on whether the system we want can be built <u>and operated</u> for the money we intend to spend, using a specific design alternative. We must consider each design alternative's life-cycle costs. We have to look at the initial system development costs, the costs to operate the system (including people costs), and the costs to maintain it (both hardware and software), including the cost of supplies associated with the system during its operation. We evaluate these costs over the expected life of the system, i.e., the amount of time we expect the system to be in use. If a system can be built for the development money that we have, but costs too much to operate and maintain, it does not have cost feasibility. Take the alternative that we assessed as the best technical solution above, the one with the most capacity. If the life-cycle costs of that alternative are beyond what we believe we can afford to invest in this solution, we can't consider it the "best" solution. Instead, we have to go with the "better" solution, if that solution is affordable.

- ***Schedule feasibility*****:** This is our judgment on whether the system we want can be built, using a particular design alternative, within our implementation time frame. There may be external factors to consider, such as whether a supplier can provide one or more necessary components on time. There are always internal factors, such as staff availability for our project and staff skills, that we must consider when assessing schedule feasibility. If a particular design alternative can't be used successfully by our deadline, we have two choices: eliminate it or change the deadline.

What we're after in assessing the technical, cost, and schedule aspects of our alternatives is to choose the alternative that is the best combination of the three. It must be technically feasible, it must have a reasonable (affordable) cost, and it must be one that we can build in the required time frame. That's the solution that our trade-off study should lead us to.

Managing Uncertainty and Risk in Our Systems

If we could accurately predict the future, it would be easy to avoid mistakes and problems. However, in real life, we need to deal with uncertainty and risk. Systems engineering focuses on three aspects of uncertainty and risk management:

- Identification
- Analysis
- Mitigation

Identification deals with ferreting out the issues and potential problems that can affect our project. Table 1 contains a set of questions that you can ask yourself, to help identify issues and potential problems that you might have. The questions are at a high level. As you answer them, other, more detailed questions will arise that you must also address.

Analysis assesses the importance of each issue and problem and determines what we know about it. One way to analyze the issues and problems is to rank-order them according to their importance and the degree of uncertainty in our knowledge about them. This organizes them into a 3x3 matrix, as shown in Figure 2.

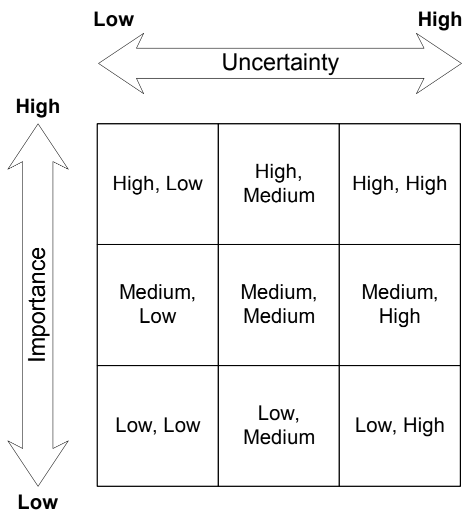


Figure 2
Assessment of Importance vs. Uncertainty

The upper right-hand cell of our matrix is the one of greatest concern. These are highly important issues and problems about which we have a high degree of uncertainty, i.e., the inverse of knowledge. The lower left-hand cell is of least significance. The seven other cells are all addressed according to whatever scheme we deem best, but generally we address the issues in each cell, moving from left to right, top to bottom, as we move between cells.

Table 1
Key Questions

Area	Key Questions
Needs Analysis	• What is wrong with the current situation? • What needs does the ITS project fill? • Have we clearly articulated the need? • Do all ITS project stakeholders have a common understanding of the project's goals and objectives?
Concept of Operations	• Is our concept consistent with any Architecture(s) with which it must interact? • Have we identified all intended users of the ITS system? • How will each intended user interact with the ITS system? • How is this different from the current situation, if at all? • Do the intended users understand their role in the ITS system? • Have we coordinated with all other agencies affected by this ITS system?
Requirements	• What specific functions will this ITS project perform? • Have we defined each function in detail? • Have we identified all system interfaces? • Are all system interfaces well defined? • Have we defined our required system performance in quantifiable terms? • Have we reviewed all requirements with stakeholders? • Have we considered system availability requirements? • Have we assessed our reliability and maintainability requirements? • What derived requirements must we validate with our customer(s)? • Have we considered what security our system needs?
System Architecture	• What are the components of the ITS (e.g., TMC, ATIS)? • How does this ITS system fit in with other ITS systems in the region? • Is there an existing regional or project architecture based on the National ITS Architecture?
Allocated Requirements	• Which components address which requirements? • Is this allocation appropriate? • Is this allocation complete? • Are there any unaddressed requirements?
Detailed Design	• Do the details meet the requirements? • Is each component buildable? • Are the interfaces satisfied? • Are the details well documented? • Do the details of the design map to all allocated requirements? • Have we built sufficient redundancy into all mission-critical components?
Implementation (includes system integration)	• What is the overall plan for building this ITS system? • If capability will be phased in, what is our overall schedule? • Can the ITS project be completed within cost and schedule? • Have we considered human factors? • Have we assessed the maintainability of the final system? • Can we re-use/integrate existing components or capabilities?
Test	• How will we know when a test is successful? • Are all mission-critical functions thoroughly tested? • What areas will we not test and why? • Have we scheduled a full end-to-end test, integrating all interfaced systems?

Area	Key Questions
System Acceptance	• Do we have clear criteria for system completion? • Have all users agreed with our completion criteria? • Will our customers be satisfied with the system? • Will we have adequate system documentation for all users and maintainers?
Operation and Maintenance	• Have we assessed the full life-cycle costs of the system, including training, operation, and maintenance? • Have we identified who will maintain the system? • Do we have the system maintainer on-board? • Do we have a schedule for upgrades and/or enhancements to this system? • What growth in demand have we planned for?

Analysis continues with an evaluation of what it will cost us, in terms of time, money, and effort, to increase our knowledge about each of these items so that we are comfortable with the degree of certainty that we have. Once we know that, we can decide how we want to invest our resources (time, money, and effort) in ***mitigation***.

Mitigation involves performing some action to gain knowledge and reduce the risk related to an item. The types of actions that we can perform include:

- Prototyping
- Modeling and simulation
- Experimentation
- Trade-off studies
- Market assessments
- Surveys and questionnaires

While that list is not comprehensive, it illustrates ways we can reduce uncertainty and mitigate risk.

Designing Quality into Our Systems

A US automobile manufacturer once had as its slogan, "Quality is Our Most Important Product." For systems engineers, that's a motto to take to heart. Systems engineering is all about building quality into systems. Consider the following definition from one of the world's most important standards-making bodies, the International Organization for Standardization (ISO), which defines quality as:

> "…the totality of features of a product or service that bear on its ability to satisfy stated or implied needs."

If we replace the phrase "product or service" with the word "system," the definition fits our needs well.

Systems engineering helps us design quality into our systems by giving us tools to address factors that can affect quality negatively. Among these are:

- **Complexity**: If a *system* is complex, it consists of many different, dissimilar parts, each of which has to interoperate with many other parts for the system to work successfully. If the *design* of a system is complex, there are many different parts, each of which has to be well understood to build the system successfully. It may not be possible to reduce the complexity of the system, but it serves us well to reduce the complexity of its design. The simpler the design of the system, the easier it will be to build.

- **Inflexibility**: An inflexible *system* is one that cannot be easily adapted to changes in its environment. An inflexible system *design* is one that cannot be easily adapted to changes in its requirements. Either type of inflexibility affects quality negatively. Without adding so much flexibility that we make the system or its design overly complex, we need to build in the ability to deal with changes. Change is inevitable but we must accommodate it. We must control how and when it is done.

- **Lack of Standardized Components**: When a system is built with one-of-a-kind components or with components built around a vendor's proprietary products, it may lack *maintainability*. Maintainability is a measure of how difficult it is to keep a system performing correctly as time passes and the system ages or requirements change. With one-of-a-kind components, if a component ceases to function, it must be re-fabricated. This can be anything from building a new hardware item that fails or reconstructing software that no longer meets requirements. If the system depends on a vendor's proprietary products, one must go back to that vendor if a component must be replaced. If the vendor is no longer in business, a new product must be integrated into the system, leading to considerable effort to ensure that the new products works with all other system components in the same way that the original product did. Ideally, one would build a system using commercially available products built using an industry standard. Then, if a component fails, a compatible product should be available from multiple sources. While it may be necessary to build some ITS systems with unique or proprietary components, more standardized system components are becoming available. A good systems engineering approach would identify these components and adapt the system design to use them.

  Another key point related to the lack of standardized components relates to the issue of system expansion or growth. If you are dealing with standard components and system growth requires you to add more of them, it's a simpler acquisition. You're buying more of a product that you can purchase from multiple sources, so there's competition and the likelihood that you'll get a reasonable price on those items. On the other hand, if system growth requires you to add more <u>nonstandard</u> components, you're faced with a more difficult acquisition. The components are probably only available from the original supplier. There's no competition, so there's no price break; you're at the mercy of the original supplier. Or worse, the original supplier may have gone out of business. This may force you to re-engineer these components and hope that the re-engineered ones work the same as, and in conjunction with, the remaining original components.

- **Reliability and Availability:** *Reliability* is a measure of a system's ability to function without error. In software, one increases reliability by finding and fixing errors in the system's programs. One technique for doing this is testing. However, it is impossible to test all possible paths that a complex program can take during its execution. Therefore, even extensive testing does not guarantee 100% reliability. You should have contingency plans in place to handle situations where the system turns out to be less reliable than you hoped. A good systems engineering approach will identify the risk areas that exist and propose reasonable contingency plans to address them. *Availability* is a measure of how much outage, i.e., times when the system (or some component) is not functioning, a system has. Availability requirements are frequently stated in terms of the number of "9s" you are trying to achieve. For example, two "9s" means 99% availability, three "9s" means 99.9% availability, and four "9s" means 99.99% availability. Since all components of a system are liable to fail at some point, the greater the availability we define as a requirement, the more backup systems or backup components we will need. This comes at a cost, which can be very high. Take a system that must operate 7 days a week, 24 hours a day. There are 8,760 hours in a 365-day year. If we call for 99% availability, the system can have only 87.6 hours of downtime during the year. With 99.9% availability, only 8.76 hours of downtime are permitted. With 99.99% availability, only .876 hours of downtime (about 53 minutes of downtime) are permitted. The other issue in determining what the requirement for system availability is lies in deciding what constitutes system availability? Do all components of the system have to be available all of the time? Can some be down without affecting the measurement of overall system availability? These issues are not trivial, particularly when there are contractual conditions involved. Systems engineering considers and answers these types of questions.

Again, the list above is not all-inclusive, just illustrative.

One place where we can begin building quality into our system is in the *Concept of Operations* (see next section). The *Concept of Operations* represents the "vision" of the system that is communicated to all "stakeholders," i.e., individuals, organizations, and groups that have an interest in seeing the ITS project succeed.

Handling Project Management Issues that Arise

One key to a successful project is a sound project plan. This means that a plan exists that has:

- Complete and comprehensive list of tasks that must be performed
- Adequate resources (labor, time, money) to accomplish all tasks

- Defined dependencies, where they exist, among tasks to show the order and precedence of task accomplishment
- Defined products or completion criteria for each task
- A method of measuring progress against plan

Having a good plan is not enough. The plan must be communicated to and understood by all of the people working on the project. Everyone who is scheduled to work on a task needs to know:

- The work product(s) of each task for which they are responsible
- When the task is scheduled to begin
- What the relationship of the task is to other tasks (i.e., on what completed effort does the task depend, what depends on the successful completion of the task)
- Who else will be working on that task
- When must the task be completed
- What resources other than people are required to perform the task

How one prepares a realistic schedule is beyond the scope of this paper. However, once the project is underway, there are several important project management functions that systems engineering can support. These include:

- Tracking
- Measuring progress
- Revising schedules when required
- Addressing obstacles to progress

Tracking involves ensuring that tasks begin and end on schedule and, if they don't, determining what caused the variance from plan. If the variance is unfavorable, i.e., the task begins or ends behind schedule, you must assess what prevents the scheduled start or completion of a task. It could be the lack of an important resource or the failure of a related task to start or end on time. Determining the root cause of the problem, however, is important if you intend to solve it. If the variance is favorable, i.e., the task starts or completes ahead of schedule, you must still assess the impact of the early start or finish. It may be possible to shift resources from a task completed early to work on efforts scheduled later in the project. This may have the overall effect of shortening the project's duration. But it is also possible that the early start or finish of a task only means that some resource is underutilized for a period, since there is nowhere else on the project where you can use the resource.

Measuring progress means monitoring the consumption of resources on each task to determine whether the product(s) being delivered is(are) consistent with the time, effort, and money being expended. It helps to have quantifiable, objective means of measuring, i.e., "metrics," available so that you can be objective rather than subjective in determining how much progress has taken place. "Percent complete" is a subjective measure unless you have a quantitative value to measure. For example, if you know you have 50 different signals to install at intersections, you can say you're 30% complete when 15 are installed. It's harder to determine when you're "30% complete" in coding a program.

Revising schedules is a re-planning effort that should be considered periodically on projects that last at least a year. Feedback on the accuracy of schedule estimates is necessary to help planners improve their planning efforts. A quarterly project review should assess whether the overall schedule for the project is still realistic, given the progress made in the previous quarter. If not, you should revise your estimates, given that you better understand what each of the uncompleted tasks requires, to reflect what work each task entails. However, make it clear that you have revised the project plan if you lengthen it. You want all of the project's stakeholders to recognize the schedule change explicitly.

Addressing obstacles to progress is important when there is a problem with starting or completing one or more tasks on schedule. Identifying the problem is not enough; you must also come up with a solution that removes the obstacle or reduces its impact on the overall project effort. The goal is to complete the project as close to the original schedule as is feasible and still deliver a quality system.

Our Approach to Illustrating Systems Engineering Concepts

In the rest of this paper, we'll cover many key topics in systems engineering, although at the "30,000 foot" level. To help make these topics clearer, we'll use the following artificial example of an ITS project to illustrate what we mean.

Metropolis and Gotham City, two cities less than 50 miles apart in the State of Bliss, are linked by Interstate Highway 105 (I-105). There are a number of ITS elements deployed between the two cities, and there is a regional Transportation Management Center (TMC) that manages these elements. The MetGo County regional planning commission envisions an upgrade to the ITS capabilities along the Metropolis-Gotham City corridor, to provide an enhanced infrastructure with the ability to handle both growth and additional functionality. The regional planning commission is working with the State of Bliss Department of Transportation (BDOT) and its Chief Traffic Engineer for the MetGo region, M. D. Mandrake. At the latest planning meeting, Chief Traffic Engineer Mandrake provided the assessment of existing capabilities and needs shown in Figure 3.

After the Chief Traffic Engineer gives this initial report, she and her staff begin work on preparing an ITS project to deal with the situation that they've assessed. We'll see how this works out in the rest of this document.

Well, that's the approach that we're going to use. So, let's get started.

Description of the Problem

"One of the major problems we have in managing our ITS elements in this region is the multiplicity of communications networks that we are using, along with the limited capacity of some of those networks. We have the following communications networks currently in use:

- Our Dynamic Message Signs (DMS) are controlled by voice grade telephone lines
- We have separate voice grade telephone lines to use for entering data into our two Highway Advisory Radio (HAR) sites
- A third set of voice grade telephone lines links five existing video detection sites along I-105
- We have an additional nine video surveillance sites along I-105 that are linked to our TMC by analog microwave

This gives us four relatively low-speed communications networks that operate independently, with a large number of connections required at our regional TMC. In addition, three of the networks are saturated and cannot handle additional communications traffic. The fourth is close to its rated capacity.

We have requests from police and fire agencies asking to share the surveillance data from our cameras. In addition, those agencies have suggested the installation of additional surveillance cameras on secondary routes feeding to and from I-105, to assist them in responding to incidents in the area. Agencies that have requested data sharing or additional camera installations are:

- Bliss State Highway Patrol
- Metropolis Police Department
- Metropolis Fire Department
- Gotham City Police
- Gotham City Fire and Rescue
- MetGo County Sheriff's Department
- MetGo County Volunteer Fire and Rescue

In addition to the upgrades for our communications networks, we need to upgrade our computers and consolidate existing software. At present, each separate system (VMS, HAR, the two video surveillance groups) is on a separate computer. There is no backup computer available for any of these systems.

We would like to put all four systems onto a single large server and have a second computer, of similar capacity, available as a backup to the main processor. If we do this, the software for each of the existing systems will require some modification to run under the operating system used by the server.

We intend to initiate this upgrade by installing a single high-capacity telecommunication network along I-105. This network will take the place of the existing four networks and will give us additional capacity. After this network is installed, we can run branch networks to each of the police and fire agencies that have asked for data sharing. We will require some additional network software to control the distribution of data among the agencies and our TMC, but we believe we can do this with commercial off-the-shelf software.

Once we have our communications upgrade completed, we will upgrade our computer facility, replacing our existing processors with newer, more powerful machines."

Figure 3
Description of the Problem

2.0 The System Life Cycle and Systems Engineering

So when and how does systems engineering fit in as you build a system. Before getting too far along, we'll define some terms. Definitions help ensure that the author clearly communicates what's meant by key terms. There are three key terms in this discussion:

- System
- System life cycle
- Systems engineering

Definitions

A ***system*** is something whose parts work together to achieve a common goal. That's a basic definition of the term, but it has all the components of a more formal definition[3]. No definition of "system" states what size a system has to be. In fact, the parts of a system can themselves be systems. When we describe an ITS system, we call it a "system of systems," i.e., each item in it is (at a high level) a system itself. For example, you can find an Advanced Traveler Information System, an Advanced Traffic Management System, an Emergency Management System, a Transit Fleet Management System, an Automated Transit Information System, and a Traffic Control System in an ITS system. Where you draw the *boundaries* of a system determines what its components are. However, the wider you draw the boundaries, the more complex the interactions of the system's parts become.

A ***system life cycle*** starts when the need for the system is conceived and ends when the system is discarded. The *duration* of that life cycle varies. Also, the way we break the life cycle into its component parts varies. Although there are many different ways of breaking a system life cycle into its component parts, we're going to use the following *stages* or *phases* in our discussion:

- ***Conception***: This begins when you first recognize the need for a system (or new system version). During this stage, you do two things: 1) perform a *needs analysis*, and 2) develop a *concept of operations*. The *needs analysis* determines what problem you are trying to solve; the *concept of operations* is a user-oriented view or "vision" of the system that you want to implement. You develop a *concept of operations* to help communicate your vision of the system to the other "stakeholders," i.e., the other agencies, organizations, and individuals whose interests the system affects. It also highlights the interfaces that the system has and ensures, through the publication and discussion of the concept of operations with others, that you have identified all interfaces. During this stage, you establish the high-level requirements for the system.

[3] Webster's *New Collegiate Dictionary* defines 'system' as "a regularly interacting or interdependent group of items forming a unified whole". The idea of working toward "a common goal" is implied.

In some depictions of the system life cycle, this stage is known by the name of one of its major products, the *concept of operations*. When we discuss a graphical depiction of the systems life cycle later on, we'll use the ***Concept of Operations*** title for this stage.

- ***Requirements analysis***: This stage allows you to determine, in a more detailed manner than you did in the *Conception* stage, what you want the system to do. If the system will replace an existing one, many requirements may be embodied in what the existing system does. However, you must *abstract* from "how" the existing system satisfies those requirements to a description of "what" the new system must do. It's also possible that the new system will have requirements not found in the existing one; after all, that may be a reason you want to replace it. This stage can run through several cycles itself, as you define, review, and refine your requirements. A key point related to this phase is that the end product must be a set of requirements on whose meaning everyone agrees.

  Frequently, as you'll see later on, this stage is divided into two major parts, each identified by the detail level of the requirements developed during each part. The two major parts are: ***High-level requirements*** and ***Detailed requirements***.

- ***Design***: This stage involves deciding "how" each requirement in the system is satisfied. It heavily involves systems engineering, since there are many trade-off decisions to make. It is also important to ensure, as the design evolves and becomes more detailed, that the design retains its internal consistency. If groups working independently on parts of the system design fail to communicate effectively, it may be difficult to make the system's pieces mesh during implementation.

  It's also common to view this stage in two major parts, ***High-level design*** and ***Detailed design***.

- ***Implementation***: This stage transforms the design into a product. It may involve building parts of the system from scratch or integrating the different pieces that you buy. However it is done, it makes the system real.

- ***Integration and Testing***: The testing stage overlaps implementation. It starts with *unit testing*, i.e., the testing of individual pieces of the system, as soon as any pieces are available. It continues with *string or integration testing*, which ensures that individual pieces work together as intended and which tests interfaces at the lowest level within the system. Last is *system testing*. This involves a full end-to-end test of the complete system and comes in two major "flavors." The first is the final integration test by the system developer (whether that's you or a contractor you hired). The second is the "acceptance" test, where the end user(s) of the system make sure that it does what it's supposed to do. The acceptance test, however, is conducted in the next stage.

- ***System Acceptance***: During this stage, you test the completed system prior to putting it into production use. Users and operators should participate in the system acceptance process. During this stage, in preparation for the system acceptance process, you must train the users and operators of the system so they understand how to use it to do their jobs.

- ***Operation and maintenance***: This should be the longest stage, the one in which you use and maintain the system for the remainder of its existence. Maintenance involves all processes that keep the system performing satisfactorily, including upgrades of equipment and software to later versions to enhance the system's performance as its volume grows. When you deal with an upgrade or major change to a system, the life cycle starts over, for that piece of the system that is being modified. It's also important to recognize that you have to keep monitoring the system's performance against its original performance requirements. This is particularly important as demand on the system increases. You may have planned for some growth in demand; you need to know when you reach that limit.

At a relatively early point in the system life cycle, you make your "build versus buy" decision. When exactly you make that decision depends on whether you intend to conduct the implementation with internal staff or with external staff, i.e., contractors. In most ITS projects, public sector agencies hire contractors to work with them on the development of the ITS system. If you plan to use contractors, you should bring them on board before you complete the requirements analysis stage. That way, the contractors can help develop and refine the requirements and there's a better chance that everyone will develop a common understanding of what the requirements mean.

The decision on whether to build or to buy is not simple. Even if you decide to "build" the system, you may buy equipment and services during its building. If you decide to "buy" the system, you still must devote resources from your organization to working with the vendor(s) who sell you the system. "Buy" decisions also involve determining what type of contract to use. For guidance on acquisitions, refer to *The Road to Successful ITS Software Acquisition, Vols. I and II* as well as other DOT guidance (see **Where to Go to Get More Help**, at the end of this document).

Systems engineering is the process by which we build quality into complex systems. It uses a set of management and technical tools to analyze problems and provide structure to projects involving system development. It focuses on ensuring that requirements are adequately defined early in the process and that the system built satisfies all defined requirements. It ensures that systems are robust yet sufficiently flexible to meet a reasonable set of changing needs during the system's life. It helps manage projects to their cost and schedule constraints and keeps realism in project cost and schedule estimates.

The key to the above definition is that systems engineering is a *process*, not just a set of tools. Let's look at how this process relates to the system life cycle we discussed earlier.

Where in the Systems Life Cycle Does System Engineering Fit?

The easy answer to the above question is "everywhere." Systems engineering should be an integral part of any system project. After all, it's the process by which you build quality into your systems. Let's go back to the Metropolis-Gotham City example to illustrate this idea.

Chief Traffic Engineer Mandrake has her marching orders. She kicks off her system effort with the first stage, *Conception*.

> **Conception.** This stage begins with a needs analysis. That there is a need is clear but she must figure out what the system should do. Asking one of the questions that we listed in Table 1, "What is the problem we're trying to solve?" is the best way to start.
>
> Chief Traffic Engineer Mandrake plans to develop a *Concept of Operations* for a system that her project will build. The Concept of Operations is a formal document that provides a user-oriented view of the proposed new system. The content of a Concept of Operations document is spelled out in a standard[4] from the Institute of Electronic and Electrical Engineers (IEEE), one the major U.S. standards bodies. Figure 4 illustrates the outline of a Concept of Operations document, as specified in that standard. Let's look at some of the sections of that document and see what Mandrake needs to include.
>
> The first few sections, *Scope* and *Referenced documents*, are mostly "boilerplate," necessary, but not exciting. The third section, *Current system or situation*, deals with what currently exists. We described that briefly on pages 13 and 14, although not as thoroughly as the Concept of Operations document might need. However, let's assume that the information on those pages is adequate and go on to the fourth section, *Justification for and nature of changes*.
>
> This section addresses the shortcomings of the current system that motivate either the development of a new one or the modification of the existing one. Let's review what these shortcomings are, as indicated in Figure 3 on pages 13 and 14:
>
> - Three of the four existing networks are saturated and cannot handle additional communications traffic. That's important because local public agencies want additional surveillance cameras that will increase the communications workload.
>
> - All four networks operate independently, thus coordinating them requires a complex switching system at the TMC. When new capabilities are added at the TMC, the system will grow more complex unless redesigned or replaced.

[4] IEEE Std 1362-1998, *IEEE Guide for Information Technology – System Definition – Concept of Operations (ConOps) Document*

- There are no backup computers for any of the existing systems. If any one computer goes down, the system that runs on it is unavailable.

- Each system runs independently on its own computer. This can be viewed

Concept of Operations Outline
Title page Revision chart Preface Table of Contents List of figures List of tables 1. Scope   1.1. Identification   1.2. Document overview   1.3. System overview 2. Referenced documents 3. Current system or situation   3.1. Background, objectives, and scope   3.2. Operational policies and constraints   3.3. Description of the current system or situation   3.4. Modes of operation for the current system or situation   3.5. User classes and other involved personnel   3.6. Support environment 4. Justification for and nature of changes   4.1. Justification of changes   4.2. Description of desired changes   4.3. Priorities among changes   4.4. Changes considered but not included   4.5. Assumptions and constraints 5. Concepts for the proposed system   5.1. Background, objectives, and scope   5.2. Operational policies and constraints   5.3. Description of the proposed system   5.4. Modes of operation   5.5. User classes and other involved personnel   5.6. Support environment 6. Operational scenarios 7. Summary of impacts   7.1. Operational impacts   7.2. Organizational impacts   7.3. Impacts during development 8. Analysis of proposed system   8.1. Summary of improvements   8.2. Disadvantages and limitations   8.3. Alternatives and trade-offs considered 9. Notes Appendices Glossary

Figure 4
Concept of Operations Outline

as a shortcoming because it means that having a backup system (as of now) for each system would require four separate backup computers.

Those shortcomings are taken from the original problem statement Mandrake provided. But she knows that there are some other operational issues that need to be addressed. The seven public safety agencies asking to share data want to manage their incident response better. To help them do this, the TMC needs to understand what data they need and when they need to get it. The TMC also needs to know which agencies to notify and transmit surveillance data to and what criteria to use in notifying them and sending the data.

To address these operational issues, Mandrake must interact with representatives of those agencies, gather data, and use that data in compiling the Concept of Operations. These interactions provide the basis for justifying the changes, determining exactly what they should be, and establishing the priority in which the changes should be implemented. During this process, Mandrake may consider some changes that she ultimately decides to forego. If so, the Concept of Operations provides a section in which to document these rejected options. And this is one of the several benefits of a Concept of Operations document. By writing down the options that she considered but rejected, along with the reasons she rejected them, Mandrake creates a tool for use in later stages of the project. Should someone resurface one of these options, or a similar approach, later on in the project, Mandrake can review the original reasons for rejecting the option and get to closure on the discussion more easily and more quickly. This is true even if additional data leads to a change in the decision.

The last subsection in the *Justification* section documents assumptions and constraints. Putting assumptions and constraints in writing makes it easier to communicate them to all interested parties. And, if any assumptions or constraints change during the project, Mandrake can go back and see what effect they had and what impact a changed assumption or constraint would have on how the system evolves.

The fifth Concept of Operations section, *Concepts for the proposed system*, gives Mandrake the opportunity to lay out the system and explain how she thinks it will work, once it's in operation. This is a key section, because it shows the intended users of the system how the new system will affect them. It's a good tool for setting or managing user expectation for the new system. Note that one subsection specifically deals with operational policies and constraints related to the new system. These might include items such as:

- Limits on the hours of operation of the system, or any part of the system
- Policies related to system access or access to secure areas of the system
- Constraints related to specific hardware that must be used

It's in this section that Mandrake addresses the operational protocols that must exist for determining which agencies get data and when and how they get it.

The sixth Concept of Operations section, *Operational scenarios*, allows Mandrake to describe specific cases of system use. Some like to use flow diagrams to illustrate operational scenarios; others like to use a narrative approach, sort of a "Day in the Life of …" approach. We've provided a narrative example operational scenario in Figure 5 below.

Perspective – MetGo Regional TMC Operator

Joe, an operator at the MetGo Regional TMC, sits at his console. The console includes a sloping display, raised/curved keyboard, proper distance from display to eye, and adjustable height for maximum comfort, all sound human ergonomics. He logs onto the operating system and runs a software application that gives him "operational" access to the systems in the TMC. Using his keyboard, pointing device, predefined operating instructions, and the TMC video wall, Joe monitors and operates the system. Throughout the day, he monitors and controls functions and systems within the TMC, which include:

- Switch any surveillance camera images available on the network to any monitor(s) or display screen(s) in the TMC
- Modify or update special event data on any MetGo County, BDOT, Metropolis, or Gotham City websites
- Retrieve Preventive Maintenance schedules for system fine tuning
- Access the TMC information database for update purposes
- Change any HAR message within the MetGo County area, selecting from an approved message list
- Access any DMS in the TMC's operating area and change messages to be displayed, selecting from an approved message list
- View traffic congestion and incident data fro both surface traffic street and freeways
- View automatic vehicle locator (AVL) data form the Metropolis and Gotham City Police Departments and the MetGo County Sheriff's Department

Alerted by the sudden decrease in vehicle speeds along Palm avenue, Joe selects the video images from Palm and Date Streets in Gotham City. He notices that the northbound lanes on Palm are backing up due to a stalled car on the outside lane. The problem is compounded by the fact that a construction crew is working next to the car, on the inside lane. Looking more closely, he sees the construction crew assisting the driver out of his car and onto the street. He switches the image to the large screen display – it looks like a medical emergency.

Joe logs the incident using an Incident Management Report window. He records information about the incident – reporting agency, contact name, phone, incident location, lanes affected, place on map, description, notification list, traffic management plan, and alternate routes. The incident report is immediately and automatically routed to the Gotham City Police Department's computer-aided dispatching (CAD) system, alerting operators who can react by dispatching police and emergency services. Additionally, the report automatically updates local DMSs (already in use for the construction activity) to warn oncoming traffic of the incident and suggest alternate routes.

Quick action addresses the medical emergency, possibly saving a life. Travelers receive information about alternates for navigating more efficiently through or around the problem area.

Figure 5
Sample Operational Scenario

The seventh section of the Concept of Operations, *Summary of inputs*, and the eighth section, *Analysis of proposed system*, recap much of the material covered in earlier sections, but in a more condensed form. The ninth section, *Notes*, is where you'd put additional material, things that don't fit anywhere else, but help the reader understand what's in the Concept of Operations.

Requirements Analysis. In developing a Concept of Operations, Mandrake looked at user needs and desires and the Concept of Operations document reflects these. However, those needs are usually stated in broad terms and need to be refined before they can guide the design of the desired system. We deal with the subject of *requirements* in more detail in another paper in this series, ***Developing Functional Requirements for ITS Projects***. We'll just cover a few points here.

For requirements to be most useful, they should be statements of <u>what</u> is desired, not descriptions of <u>how</u> the need should be satisfied. They also have to be clear, complete, and correct. It is the job of the systems engineer to ensure that these last three attributes are satisfied in the statement of requirements. The systems engineer must review the statement of each requirement and look for ambiguities or possible misunderstandings that can arise from the way it is described. One way to eliminate ambiguity is to conduct a System Requirements Review. A System Requirements Review brings in representatives from each stakeholder and walks them through your understanding of each requirement. The stakeholders have to agree: 1) with your interpretation of the written requirement, and 2) that this interpretation meets their needs. Once everyone agrees on the meaning of the requirements, the next stage can begin. (Note: Once the builder/integrator of the system is selected, the same kind of system requirements review should take place, only now with representatives of the builder/integrator present. Otherwise, the builder/integrator may not have the same understanding as the other stakeholders of each requirement's meaning.)

Don't freeze requirements! Control them!

Once you have completed the development of your system's requirements, the natural tendency is to "freeze" the requirements, i.e., assume that they are fixed and cannot be changed. In reality, freezing requirements is counter-productive. Since systems don't happen overnight, it is likely there will be at least one legitimate major change in the system's requirements before the system goes into production use. Instead of freezing requirements, the best systems engineering practice controls changes to requirements through Configuration Management. We'll discuss Configuration Management later.

Design. During this stage, you will make some high-level decisions about how you expect to implement the system. For example, you may decide (or at least narrow the decision on) where to place the Traffic Management Center you plan as part of this system. Some key systems engineering activities in this stage are to identify what trade-off decisions are needed and to conduct the trade-off studies to provide information on which to base these decisions. As an example of a trade-off study, let's say Chief Traffic Engineer Mandrake's ITS plan includes Automated Vehicle Identification (AVI) tags to measure and report travel times at congested sites among the major arteries between Metropolis and Gotham City. Since the type of tag you use has an effect on the specifics of system design, she must assess the types available to her against her selection criteria. Table 2 is an example of the results that might occur from a trade-off study, with criteria, weights assigned to criteria, and relative ratings assigned to each factor assessed. The trade-off studies clarify the *Concept of Operations* for the system.

Table 2
AVI Tag Trade-Off Study Matrix

		Type of Tag								
		IVRT[5]			**Toll**			**Beam**		
Criteria	**Weight**	**Characteristic**	**Ranking**	**Score**	**Characteristic**	**Ranking**	**Score**	**Characteristic**	**Ranking**	**Score**
Shelf Life (Recurring cost is most important	0.3	Battery powered, 2 to 5 years	2	0.6	Battery powered, 5 years	5	1.5	Beam-powered, unlimited	10	3.0
Minimum Quantity (Don't want to order more tags than can be distributed)	0.2	Large quantities required to initiate an order	2	0.4	Low	10	2.0	Low	10	2.0
Cost (Initial cost is less important than recurring cost)	0.2	Low, tags contain no intelligence or protective packaging	10	2.0	High, tags contain intelligence to support toll-collection transactions	2	0.4	Medium, tags are passive and contain no intelligence	5	1.0
Adhesion Method	0.1	Sticker with adhesive backing attached to windshield	5	0-.5	Velcro backing attached to Velcro on windshield	8	0.8	Velcro backing attached to Velcro on windshield	8	0.8
Distribution Method (Has to consider that two major jurisdictions are involved)	0.2	Combined with state vehicle registration sticker that is renewed each year	5	1.0	Voluntary	10	2.0	Voluntary	10	2.0
Total	1.0			4.5			6.7			8.8

[5] Intelligent Vehicle Registration Tag (IVRT)

As we flesh out the details of the system in our design, we use system engineering activities to guide decisions on how to satisfy our requirements. For example, we use modeling and simulations to test assumptions about a design choice's ability to meet our performance requirements. These performance requirements may be either an ability to handle a specified range of data volume over a specified time or an ability to process input and return a response in a specific window of time.

In this stage, we'll also again make use of reviews. Design reviews generally fall into two major categories: Preliminary Design Reviews and Critical Design Reviews. You conduct the Preliminary Design Review (PDR) on each component of the system. The PDR: 1) assesses the progress, technical adequacy, and risk mitigation involved in the selected design approach; 2) determines whether the item being reviewed is compatible with defined performance requirements; 3) estimates the degree of technical risk remaining; and 4) verifies the existence and compatibility of all interfaces involved. You conduct the Critical Design Review (CDR) when the design for each component is complete. The CDR: 1) ensures that the item under review satisfies all requirements allocated to it; 2) ensures that the item is compatible with all other items in the system; and 3) assesses whether there is any remaining risk associated with the item. You also conduct trade-off studies here as part of your risk mitigation approach, although the trade-offs are more detailed.

Implementation. This is the stage where all components of the system are either built or brought together into the overall system. Your systems engineering activities focus on ensuring that what gets built matches what you designed. In the implementation stage, systems engineering is synonymous with quality engineering. When you are applying standards as part of your design approach, systems engineering activities focus on ensuring that the developers adhere to the standard (or justify a waiver from the standard, if necessary).

Hardware development is done on some ITS projects, at least in some States, but hardware development is not always done. When your project is developing hardware, you may want to consider some of the issues associated with *configuration management* for hardware. One place to start is another monograph in this series, ***The ITS Guide to Configuration Management***. That monograph covers configuration management topics for both hardware and software.

Integration and Testing. You design your systems engineering activities to ensure that all relevant tests are included in the test effort. *Functional* testing looks at each of the functions or operations that the system performs and validates that each function or operation satisfies its requirement(s). *Coverage* testing looks at the different logical or physical paths through which data travels and ensures that at least the mission-critical ones are tested. (It is usually impossible to test all paths in a complex system.) *Volume* or *stress* testing looks at the system's ability to handle the peak load expected during its operation. The system may operate in "degraded" mode at peak conditions; this is only acceptable if that's what you planned for it. *Integration* or *"end-to-end"* testing looks at the system's ability to interface all of its components with each other and to interface itself with all other systems with which it shares data. For each type of test that you plan to conduct here, you devote your systems engineering activities to creating complete and comprehensive test sets that effectively exercise the system and its components. In addition, your systems engineers analyze the test results and assess the degree to which the system satisfies its requirements.

This stage also deals with the actual installation of the system in pre-production use. It includes preparing all of the instruction manuals for the system and all training materials, and training all users of the system. The training conducted here is the initial system training. Additional training is needed during the next stage as new users come on board and the system is enhanced with new features. Your systems engineering activities during this stage focus on ensuring that the system is fielded so it does not disrupt the organization's critical business functions. For example, if you're installing new sensors throughout an existing road network, you recognize it may be necessary to close certain travel lanes while installing them. However, these closures should take place during low usage times on the affected roadways.

System Acceptance. This is the final step before putting the system into operation and maintenance. System acceptance implies that the operators and users take ownership of the system. System engineering activities here focus on ensuring that all of the needs of the users and operators have been met. Many of the systems engineering activities relate to developing a sound system acceptance test process, one that is both fair to the developer and ensures the needs of the system's new owner(s) are met.

Operation and Maintenance. The primary focus of systems engineering activities in this stage is performance. If problems arise with the system during operation, you want to *detect* the problem, *diagnose* its cause, and *deliver* a solution with minimal *delay*. It is difficult to predict the system engineering activities here, unless they deal with analysis for possible enhancements. However, one key activity that does normally occur here is performance measurement. The systems engineer wants to ensure that the system is meeting its performance goals, established during the initial project stages. Of course, the system's performance goals have to be quantifiable or you can't measure them.

Before we go too much further, let's distinguish between different types of performance measures one can apply. One type of performance measure that's been used in discussions of ITS systems relates to the performance of the *transportation system* itself, as influenced by the use of ITS systems to improve its efficiency. One performance measure frequently discussed here is travel time/delay. For example, a state DOT could measure total person-delay time (delay time per incident times number of persons delayed by the incident) or per-

person delay time (average amount of delay experienced by all persons affected by an incident). You can have two incidents with the same total person-delay time (e.g., 100 people delayed for two hours versus 2,000 people delayed an average of 6 minutes), but with a very different impact on the people affected. Another type of performance measure relates to the performance of the *ITS system* itself, or its components. In this case, you might measure actual network carrying capacity against planned capacity or actual system throughput against planned system throughput. You could perform both types of performance measurement during the operations and maintenance stage. But, since collecting and analyzing the data required for either type of performance measurement is expensive, you should only do this if you seriously plan to use the information you get as the basis for decision-making.

A good way to see how systems engineering and the system life cycle interact is to look at the life-cycle as what's known as a "V" model or diagram of the system life cycle. The "V" (or "VEE") model is a way of relating the different stages in the system life cycle to one another. Figure 6 illustrates the "V" model.

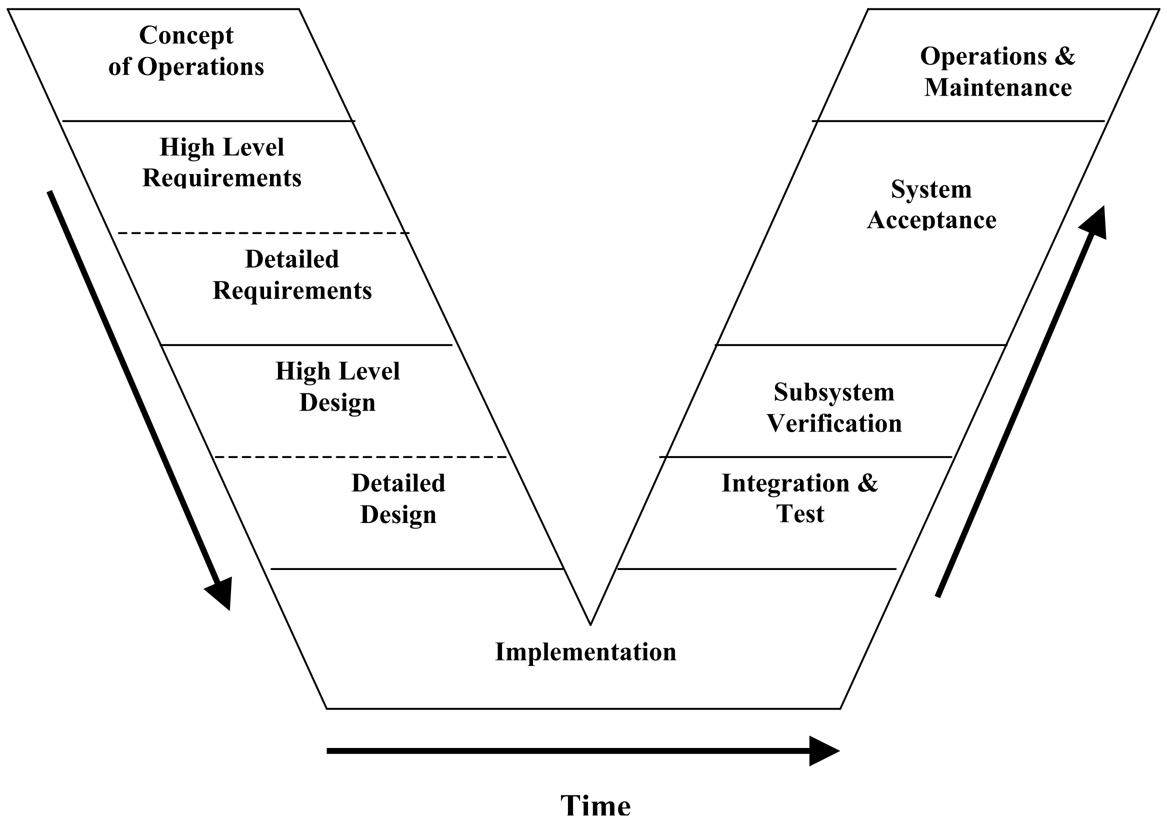


Figure 6
"V" Diagram

One reason for using the "V" model is to emphasize the importance of evaluation in all stages of a system project. In the early stages of the system life cycle (the left leg of the

"V" model), you are using mostly inspection and analysis as your evaluation tools. In the later stages of the system life cycle (the right leg of the "V" model), the primary evaluation tool is testing. Regardless of which leg of the "V" model you're on, you're interleaving evaluation efforts with system development activities.

Another reason for using the "V" model is to show an explicit relationship between work done on each side of the "V." Let's look at the relationships between the two legs of the "V" model to show what this means.

> **Concept of Operations vs. Operations and Maintenance.** During the Conception or Concept of Operations stage of the system life cycle, you create a *Concept of Operations* for the system. Concept of Operations is the term used in the "V" model for what we earlier called the Conception stage. The Concept of Operations describes how the system will work once it's built. Therefore, it relates directly to the Operation and Maintenance stage, during which the system's Concept of Operations is realized.
>
> Looking at the explicit relationship between the two stages helps the systems engineer focus, at the beginning of the project, on issues associated with keeping the system in operation and effectively maintained once it's built. This long-range perspective is important in systems engineering.
>
> **Requirements Analysis vs. System Verification.** The Requirements Analysis stage isn't explicitly shown in the "V" diagram, but it covers what the "V" shows as "High-Level Requirements" and "Detailed Requirements," since those are the products of the Requirements Analysis stage. In the Requirements Analysis stage, you determine what the system has to do, when it has to do it, and how well it has to perform. In addition, the "V" diagram calls out the System Acceptance process as "System Verification." They mean the same thing. In System Verification, you determine whether the system you built satisfies all system requirements. Satisfying those requirements involves two different approaches:
>
> - Verification – ensuring that all functions implemented in the system have been implemented correctly
>
> - Validation – ensuring that the desired functions have been implemented in the delivered system
>
> Another way of describing *verification* and *validation* is that verification is "building the system right" and validation is "building the right system."
>
> It's important to recognize that you should be able to trace every requirement in the system to the system component that satisfies it. One way to do this is through a *requirements traceability matrix*, which we discuss in Chapter 3.

System Design vs. Integration and Testing. The "V" diagram in Figure 6 actually breaks this down into two components. System Design is subdivided into "High-Level Design" and "Detailed Design." Directly across from "High-Level Design" is "Subsystem Verification," which is a reasonable correlation. Part of what's done in the High-Level Design activity is to break the system down into its subsystems, each of which is assigned a major functional area of the overall system. In Subsystem Verification, you're integrating all of the components of the subsystem and testing the subsystem(s) as units. It's the appropriate way to look at how the two stages are correlated.

"Detailed Design" is directly across from the "Integration and Test" component. Detailed Design relates to the definition of the complete system, the final design of all of the elements, at the level necessary to build it bottom-up. The Integration and Test portion of the "V" is the start up on the leg where you bring all of the pieces together. During this stage, you'll test individual components as "units" and begin testing the individual units that interact with one another, preparing to fit them all together into individual subsystems.

The design of a system, in part, allocates functions to be performed by specific system components. In addition, during design, we decide <u>how</u> we're going to implement those functions. That may involve developing *algorithms* (i.e., specific, detailed instructions on how to perform a function) or developing *interfaces* that allow devices in our system to interoperate. The integration and testing of a system involves ensuring, at all levels of the system, that each piece works as it should and that all pieces successfully and accurately interact.

The design stage is the final *decomposition* stage of the decomposition leg of system development. It's the stage where you establish, at the most detailed level, your plan for how you will accomplish the work of the system. The integration and testing stage is the first of the major *re-composition* stages, where you begin assembling the pieces of the system into an integrated whole.

Implementation. This stage transforms the design of a system into actual products. It transitions the work of system development from the conceptual level to the physical level. It is an extremely critical stage because, once the system concepts are embodied in the system products, making changes becomes much more expensive. Table 3 illustrates the experience software development organizations have had with the relative cost of making changes in software-intensive systems. By their nature, ITS systems are software-intensive.

One activity associated with systems projects that doesn't fall neatly into only one stage is deciding when and how you're going to acquire the system or its components. This is often categorized as the "make vs. buy" decision, although it's rarely that simple. Let's look at how systems engineering fits in with making the acquisition decision.

Table 3
Comparison of Relative Costs
By System Life Cycle Stage

System Stage	Minimum Cost	Maximum Cost
Requirements Analysis	$1	$1
Design	$3	$6
Implementation	$10	$40
Integration and Testing	$30	$70
Operation and Maintenance	$40	$1,000

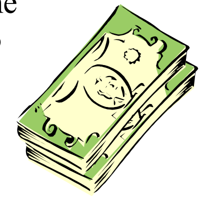
Deciding on the Acquisition Method.

When you're deciding whether to "make" or to "buy" the system, the decision is rarely straightforward. But there are several key systems engineering activities that take place here.

One is developing alternative ways, at a high level, of satisfying the system's requirements. You need to define what alternatives you'll consider in estimating a life cycle cost estimate for the system. You may have defined your alternatives in an earlier stage. If so, you can now look at the life cycle costs of each alternative as a method of deciding which one is the most feasible. However, if you haven't yet defined alternative ways of addressing system requirements, this is the point at which you should do it. The life cycle of a system lasts from the time you conceive of a need for it to when the system is discarded. Some of the types of alternatives that you may consider include:

- What types of communication systems you will use
- What types of data you will store and how
- What backup systems you will employ and where you will house them

The manner in which you answer each question will have an impact on the overall costs of the system.



In the early stages of the system life cycle, it's relatively easy to recognize some costs, namely those involved in getting to a fielded system. However, it is also easy to overlook what are probably the more significant costs of the system -- its operational and maintenance costs once it is fielded. It is particularly easy to overlook the maintenance costs of the software associated with the system, since software is not what we normally think of as "tangible." It's also easy to overlook staffing costs for the new system. In its early days, automation was touted as a way to reduce costs by eliminating labor. Machines would do the work of people. We got past that point many years ago. Now, infusions of technology tend to increase labor costs, because they require smarter, better-trained workers. ITS systems are no different.

If you don't consider early on how your staffing costs will change once the new system is implemented, there's trouble potentially waiting for you.

Another key systems engineering activity associated with the *Acquisition Method Decision* is determining what systems engineering activities you will assign to a contractor, if you use a contractor to build or integrate the system. Once hired, the contractor (or contractors) is a stakeholder, just like any other stakeholder. The contractor also wants to see the system succeed because his reputation is on the line. Some systems engineering activities are best done by the contractor, because the contractor may have more or deeper expertise in certain areas. You must allocate those activities and make clear to the contractor, in a statement of work, which ones he is responsible for.

How Does Systems Engineering Relate to Project Management?

A good project management plan will include a section devoted to systems engineering. This section, called the Systems Engineering Master Plan (SEMP), describes how you organize to conduct systems engineering activities and what systems engineering activities you plan to conduct during each stage of your ITS project. In the section above, we were mostly concerned with technical activities. Let's look at some management activities associated with systems engineering. These activities will fall into four major categories:

- Project Planning and Control
- Technical Planning
- Technical Audits and Reviews
- Cost Estimation

We'll cover each of these in more detail below.

Project Planning and Control. There are many tools available to help a project manager develop a detailed project plan and schedule. All of them assume that the project manager knows, at a sufficient level of detail, what tasks are necessary to complete the ITS project successfully. Let's assume that the project manager can successfully develop a detailed plan that includes all dependencies among tasks, where all tasks have a reasonable duration and resources assigned to them, and that is communicated to everyone responsible for completing a task within the project. Given that, we'll focus on the issue of *control*.

Control in a program involves two key ideas: 1) monitoring program activities to determine the status of tasks underway at any given time; and 2) initiating corrective action as required to overcome any problems impeding satisfactory progress. To monitor program activities, the program manager must set up a feedback mechanism that provides accurate and timely status information. This feedback mechanism can be as simple as a daily status review, lasting for perhaps an hour, with all key managers where each reports on the status of tasks for which he or she is responsible. At this time, each manager should discuss any items impeding progress on his or her tasks. It is critical, however, that these sessions not become faultfinding or finger-pointing sessions. If this occurs, it cuts off the open flow of information necessary for successful program control and participants hunker down into defensive postures rather than working together to find solutions.

Initiate corrective action if a problem occurs

No matter what mechanism the program manager uses to get accurate status information, the next step, initiating corrective action, is essential. Determining what corrective action will succeed entails finding the root cause of the problem impeding progress. If the problem is a technical one, finding the root cause may involve significant technical analysis of the symptoms. Solving it may require analyzing different technical solutions to decide which is most effective.

Systems engineering is involved in three key ways in the *control* activity. First, in defining reasonable measures of progress against tasks so that status can be measured and reported as objectively as possible. Second, in developing an information system that captures the objective data with which to measure progress so managers can effectively report on their status. Third, in analyzing problems and developing recommendations, for the program manager's assessment and approval, on how to solve them.

Technical Planning. Technical planning primarily involves how you set up your systems engineering organization and what activities you plan to undertake in each stage of the ITS project. This information is in the SEMP section of the overall program plan. In addition to the overall technical plan covered in the SEMP, you must decide what tools to use in your systems engineering activities. For example, if you plan to use simulation and modeling as a tool during any stage of the ITS project, you must decide what simulation and modeling package or language to use and on what physical computer platform to run it. If you plan to use automated testing tools during the testing stage of the project, you must decide what specific tools you want to use and, if you don't already have them, acquire them and make sure that you train people in their use.

You also must decide what specific skills you must have in your systems engineering team. Since the major focus of your systems engineering efforts are to increase the quality of the system by reducing the risk and uncertainty associated with the project, your team needs to have technical strength and depth in those areas that you deem the most uncertain (thus, the riskiest).

The third part of technical planning is deciding how you will allocate systems engineering activities between government and contractor organizations. If the government organizations involved have little systems engineering expertise, you may decide to allocate all of these activities to your prime contractor. Another option is to hire a separate, independent systems engineering contractor, who reports directly to you, not through the prime contractor, to perform the government's share of systems engineering activities. Having a separate, independent contractor is often the more expensive[6] of the two options. However, many practitioners believe it is the better option. It provides the government project manager with a check and balance system on the prime contractor. But remember, it also requires cooperation from the prime contractor to be effective.

Technical Audits and Reviews. We've mentioned two key types of technical reviews that you should conduct during your ITS project, the PDR and the CDR. The purpose of technical audits and reviews is to assess the degree of completion of technical effort before proceeding beyond critical points in the project, e.g., key project milestones. These technical audits and reviews should be spelled out in the SEMP, and clearly understood by all parties who participate in them. You must define what type of audit or review will be conducted, when it will be conducted, by whom it will be conducted, on what it will be conducted, and what effect its success or failure will have on the next steps of the project. Reviews and audits should be formal and should be documented as part of the project's audit trail.

Refine Cost Estimates as You Learn More

Cost Estimation. Estimating life-cycle costs is a management function. It also requires *domain knowledge* to know what kinds of costs are not immediately evident. *Domain knowledge* is a critical skill that your systems engineers must have. If they don't have a deep understanding of transportation systems, they may overlook key cost elements that should be considered. Your responsibility as a program manager is to ensure that you have skilled people estimating system life-cycle costs.

There are some costs that may be difficult to estimate. If you are having hardware built for the first time, there are no historical costs to use in preparing estimates. However, there are parametric costing models that can be used. A parametric costing model is one that estimates costs based on specific values (parameters) that you insert into the model. Systems engineers with experience in parametric costing can use such models as PRICE[7] to estimate the costs of hardware components that have not yet been built. There are also parametric models that are used to estimate software costs, another cost element that is notoriously difficult to estimate accurately. If possible, develop a range of estimates rather than a single cost estimate. When a

[6] From the project point of view, adding a separate, independent systems engineering contractor is more expense. This is a short-term view, however. From a systems viewpoint, if adding that contractor increases the <u>quality</u> of the system, it should be easier to maintain the system throughout its life. This means lower costs over the long-term.
[7] PRICE is a parametric cost estimation model from Lockheed-Martin PRICE Systems used to estimate the cost of different types of equipment. The model uses cost estimation relationships (CERs) that use product characteristics to generate cost estimates. CERs are determined by statistically analyzing completed projects where product characteristics and project information are known, or are developed using expert judgment. The version of the PRICE model for software is called PRICE-S. The PRICE-S model uses Source Lines of Code (SLOC) as its major input.

single number is given for a project’s cost, that number becomes a ceiling that is difficult to exceed. Since cost estimates are just educated guesses, it is better to give yourself some latitude. As you complete each stage of the project, you can refine your cost estimates and provide a more realistic range of costs. It is unlikely that you can do this with great accuracy at the outset of the project. To protect yourself, you have to set aside contingency funds as part of your project budget.

3.0 Elements of Systems Engineering

System Engineering Standards

There are a number of systems engineering standards that describe how to establish a systems engineering capability within an organization or a project.[8] The Software Engineering Institute (SEI) has created a Systems Engineering Capability Maturity Model^{SM} that describes what any organization should have in its systems engineering process, without specifying any particular process. It is an excellent reference model for systems engineering practices. Although the SEI, an organization created to improve software engineering capabilities in the Defense industry, developed this model, it is applicable to more than just software and to more than just Defense programs.

Systems Engineering Capability Maturity Model (SE-CMM)

The SE-CMM looks at two key dimensions of systems engineering, the process dimension and the capability dimension. The process dimension focuses on what the organization does and lists basic processes needed for good systems engineering. This dimension is important to us. The capability dimension focuses on *how well* an organization performs the processes. How well an organization does systems engineering is important when the organization is a system development organization. A government ITS project manager isn't necessarily involved in determining or affecting how well his or her own organization (a public sector agency) does systems engineering, but he or she is interested in how well the organization doing system development on his or her project performs systems engineering.

Version 1.1 of the SE-CMM lists 18 process areas or practices that make up good systems engineering. These processes are categorized into three groups: organizational, project, and engineering. Table 4 below shows what processes fall into each group with the process areas listed alphabetically within each group. Although the processes are listed alphabetically, some processes normally precede others. For example, one has to plan technical effort before one can effectively monitor and control it. But the planning effort could take place many times during the project's life cycle.

The *Organization* processes create the business infrastructure that supports systems engineering and are performed primarily at the organizational level. The *Project*

[8] The Electronics Industry Alliance (EIA) published its systems engineering standard, EIA 632, in 1994. It then joined forces with the American National Standards Institute (ANSI) to make this a US standard. The revised standard, ANSI/EIA 632, was approved for publication in January 1999. In addition, the International Organization for Standardization (ISO) is developing its own systems engineering standard, ISO 15288. The Institute of Electronics and Electrical Engineering (IEEE) has established a systems engineering standard, IEEE 1220, as one of its standards. These standards are mostly based on the military standard for systems engineering, MIL-STD-499 (1969) and MIL-STD-499A (1974). A third version of this military standard, MIL-STD-499B, was prepared in 1994, but never issued. Instead, the DOD opted to adopt standards developed by standards-making bodies, such as EIA, IEEE, and ISO. MIL-STD-499B was, however, the starting point for the EIA/IS 632 standard.

processes focus on the technical management infrastructure needed to develop a system successfully. And the *Engineering* processes concentrate on the technical and engineering aspects of developing a system.

Table 4[9]
SE-CMM Process Area Groupings

Organization	Project	Engineering
• Coordinate with Suppliers • Define Organization's Systems Engineering Process • Improve Organization's Systems Engineering Processes • Manage Product Line Evolution • Manage Systems Engineering Support Environment • Provide Ongoing Knowledge and Skills	• Ensure Quality • Manage Configurations • Manage Risk • Monitor and Control Technical Effort • Plan Technical Effort	• Analyze Candidate Solutions • Derive and Allocate Requirements • Evolve System Architecture • Integrate Disciplines • Integrate System • Understand Customer Needs and Expectations • Verify and Validate System

Organization Process Areas

Let's start with the process areas in the *Organization* grouping. These process areas (see Table 4) are normally performed by the organization (company, agency) in which the project resides.

An ITS project manager may have little direct involvement with most of these process areas. However, there are two process areas that are important at the project level: Coordinate with Suppliers and Manage Systems Engineering Environment. Let's look at these in a little more detail.

Coordinate with Suppliers. There are several base practices that are logically part of government procurement processes. These include:

- Choose suppliers in accordance with the Analyze Candidate Solutions process area
- Identify needed system components or services that must be provided by other/outside organizations
- Identify suppliers that have shown expertise in the identified areas

[9] Based on Table 2-4, SE-CMM, V1.1

- Provide to suppliers the needs, expectations, and measures of effectiveness held by the organization for the system components or services that are to be delivered
- Maintain timely two-way communication with suppliers

An ITS program manager uses these base practices when deciding what vendor(s) will supply products to his or her ITS project. They are all practices used in a Request for Proposals or Request for Quotes procurement. These base practices are not just performed in a procurement process, however. You should continue timely two-way communication with any supplier after that supplier is on board, to ensure that you both clearly understand what is being supplied and what is being bought.

Manage Systems Engineering Support Environment. The systems engineering support environment on your project is the one that matters to you. The project's systems engineering support environment could include:

- Computing resources devoted to systems engineering activities (e.g., prototyping, simulation, modeling)
- Communications channels on the project
- Analysis methods you specify for the project
- Systems engineering simulation tools
- Software productivity tools, such as computer assisted software engineering (CASE) tools
- Proprietary systems engineering tools you have acquired (or your organization has developed)

If a contractor performs the primary systems engineering support on your project, you want to know how the contractor manages his systems engineering support environment. You may request this information from the bidders who respond to your RFP.

We'll pass on the other *Organization* process areas and move to the *Project* ones.

Project Process Areas

There are logical relationships among the five *Project* process areas (see Table 4), which we will point out as part of the discussion.

Ensure Quality. This base practice and **Manage Risk** are related since you ensure quality by reducing the risk and uncertainty in your design and your implementation. Another way to improve quality is to inspect the products you are incorporating in your system. Inspection is done both with physical products and with software. Good software engineering practice includes the concept of code "walkthroughs." "Walkthroughs" are when a programmer convenes a group of knowledgeable programmers and reviews a section of his or her code, at a very detailed level, with them. The programmer whose code is reviewed explains what he or she is trying to accomplish and the reviewers assess whether the code accomplishes that objective effectively.

Reviews can be done on any product. A good practice is to review designs with stakeholders to ensure that they concur the design accomplishes what they expect. Mockups of reports (or actual user interfaces) are a good tool to use in a review. Letting a user get a feel for the actual look of a product helps the user identify what's good and what's bad about the product.

Manage Configurations. Configuration management is a critical base practice in systems engineering because you use it both to define the baseline version of the system or a system component and to control changes to the baseline. Why is baseline control so important? Take an example from *Computerworld*, which contained an article[10] describing problems that Philadelphia, PA and Orlando, FL had with new systems to pay teachers. The systems were issuing paychecks to dead and retired teachers and not paying active ones. Although the program managers in both cities said the problems were caused by "faulty data entry," a comment by one of the managers is very telling. He said there were data quality problems when payroll clerks entered data directly into his new system due, in part, "to changes made to the new system after the workers were trained." In other words, they trained their staff on one configuration and implemented a different one. Good systems engineering would have prevented this.

Manage Risk. One risk that you should consider and manage is the risk that some critical factor will go wrong, e.g., deliveries of key products are late, a key individual is lost while the project is underway, some external event with a negative impact on the project occurs. You assess the likelihood that an event will occur, determine what its cost will be, and determine what the cost of mitigating it is. If the cost to mitigate the event is greater than the negative cost if the event occurs, you ignore it. If you can mitigate the risk at a reasonable cost, this is good project insurance.

There are also technical risks associated with the project. Technical risks include: interfaces not working as expected, software failing to perform its expected function, and different communication systems failing to interoperate. Your systems engineering process must consider these risks and plan to mitigate them. For example, you may prototype the interface to determine if it functions as expected. Or you may test the communication systems early on, to determine if there are interoperability problems.

Monitor and Control Technical Effort. One way to monitor technical effort is to establish clear *exit criteria* for each task in the project. Exit criteria are criteria that clearly and uniquely define when a task is complete. They are effective in monitoring project task status since they provide a clear and unambiguous way of determining if the task is done. The monitoring scheme for your project should

[10] "Payroll Systems Flunk First Exams," *Computerworld*, October 25, 1999

tell you when a task may not be completed on schedule or when it is consuming more resources than you planned for it. Either situation is a warning of a problem you should resolve. If the task is either on the *critical path* or one that will become part of the *critical path* if delayed sufficiently, you must resolve the problem affecting it by determining the problem's root cause and correcting it. If the problem cannot be fixed, you assess its impact on the overall project. For example, a delay in a critical path task delays the project's overall completion; a task that runs over budget may lead to an overall project overrun.

Plan Technical Effort. One technique used in planning technical effort on ITS projects is the Work Breakdown Structure. A Work Breakdown Structure is a hierarchical depiction of a project's planned work, with each Work Breakdown Structure element defining a product to be developed. Figure 7 depicts a sample Work Breakdown Structure for a project named "ORBITS" and only shows one subsystem at any level of detail.

You assign resources, duration, and costs at the lowest level in a Work Breakdown Structure. The cost of the element at the next higher level is the sum of the costs for all of its sub-elements. Thus, if Work Breakdown Structure element 1.1.1.1 costs $10,000 and 1.1.1.2 costs $20,000, and 1.1.1.3 costs $50,000, Work Breakdown Structure element 1.1.1 costs $80,000. Estimating the resource requirements, the expected duration of the effort, and the costs of a Work Breakdown Structure element is essential in a good technical plan.

In this process area, you determine the technical parameters for your project. Technical parameters include:

- System response time
- Expected system processing volumes
- Required network bandwidth for sensor systems
- Peak traffic volumes the system will manage
- Emergency vehicle response time ranges

Establishing the technical parameters and their threshold values gives you a way to measure how well the system meets its performance requirements. If lowest levels in a Work Breakdown Structure have clear exit criteria, you can use a Work Breakdown Structure to measure a project's earned value. (Appendix A discusses earned value.)

Engineering Process Areas

The last set of process areas (see Table 4) is in the *Engineering* group. Although the SE-CMM does not specify an order for the process areas, we'll consider them in the following order:

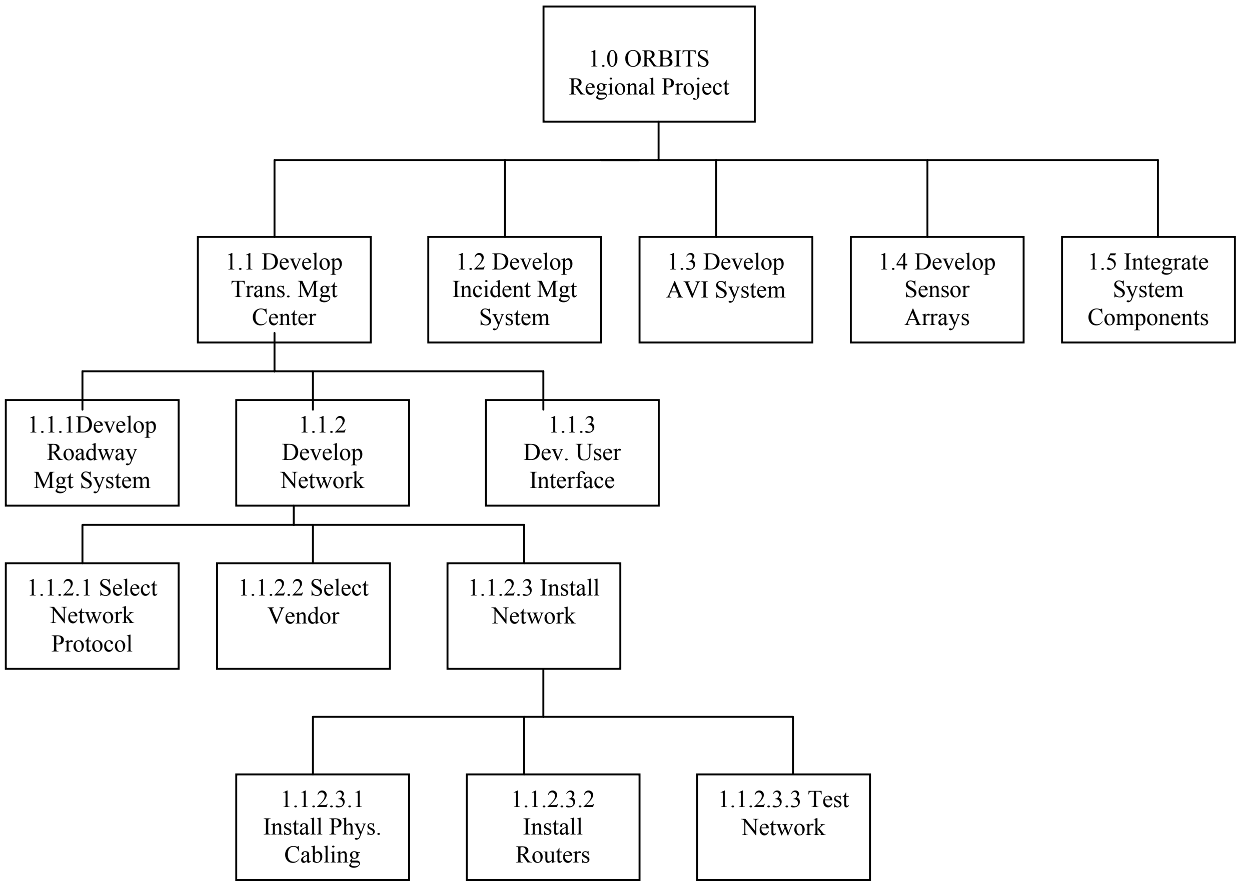

1.0 ORBITS Regional Project
1.1 Develop Trans. Mgt Center
1.2 Develop Incident Mgt System
1.3 Develop AVI System
1.4 Develop Sensor Arrays
1.5 Integrate System Components
1.1.1Develop Roadway Mgt System
1.1.2 Develop Network
1.1.3 Dev. User Interface
1.1.2.1 Select Network Protocol
1.1.2.2 Select Vendor
1.1.2.3 Install Network
1.1.2.3.1 Install Phys. Cabling
1.1.2.3.2 Install Routers
1.1.2.3.3 Test Network

Figure 7
Sample Work
Breakdown Structure

1. Understand Customer Needs and Expectations
2. Derive and Allocate Requirements
3. Analyze Candidate Solutions
4. Evolve System Architecture
5. Integrate System
6. Verify and Validate System

We've left the **Integrate Disciplines** process area out of that list. Let's discuss it first and then take the others in order.

Integrate Disciplines. In any system development project, there are "specialty[11]" engineering disciplines needed during the life of the project. Such "specialty" disciplines could include:

- Network engineering
- Aerodynamic engineering
- Nuclear engineering
- Human factors engineering
- Computer security
- Software engineering

In an ITS project, "specialty" engineering disciplines include civil engineering and transportation engineering. While not relevant areas in all systems engineering efforts, ITS projects usually require them. Another discipline that may be needed on an ITS project is transportation planning. While transportation planners may not need to be involved throughout the project, you may consider involving them during the initial planning stages of the project, to ensure that you consider a broad view of the transportation needs that the ITS project could affect. Your responsibility is to identify the engineering or technical disciplines required, get representatives from each discipline on the project team, determine how to use all the disciplines effectively on the project, and manage the interdisciplinary efforts to accomplish your project's goals.

Understand Customer Needs and Expectations. Understanding what the customer needs, wants, and expects is the starting point for your project. You should understand them to develop a reasonable concept of operations for the system you are going to build. These needs and expectations are the source of your customer's system requirements.

The customer's needs and expectations may not be clearly articulated when you start your effort. Your job (or the system engineer's job, if that's not you) is to find out what they are and to document them so that the customer can understand and agree on them. In particular, you must determine the technical performance measures for the system. Technical performance measures are quantitative values used to track the progress of your design and development to determine how well you meet the system's performance requirements. Response time (as perceived by the user) is a frequent technical performance measure. If you are delivering traffic status information to the public through a web site, you must ensure that someone looking for that information can get a response within a reasonable time, or that individual will stop looking for information there. If your web server can't handle the volume of queries it receives, the volume of queries will drop, which isn't a good sign.

One output of this process is the Concept of Operations document, which describes what the user wants the system to do and how the user expects it to operate. There

[11] "Specialty" engineering disciplines, in this context, are those engineering areas used in some, but not necessarily all, systems engineering efforts.

will be user requirements, expressed as "needs," in the Concept of Operations document, but they'll be high-level ones and will need further detail. The Concept of Operations document should identify the key users needs, as these are critical to success or failure.

Derive and Allocate Requirements. This process area logically follows the ***Understand Customer Needs and Expectations*** process area. The whole point of analyzing requirements is to get to a definition of the specific functions that a system must perform and to a specification of how well the system must perform these functions. This lets you design a system to meet these requirements. Usually, a high-level requirement (or "need") is stated too broadly to serve as an adequate design guide. Thus, we have to refine each requirement, get more precise in what we mean. Take the following high-level requirements statement: *The TMC operator shall control video surveillance cameras from the operator's console*. That's a reasonably clear statement of need, isn't it?

But what does "control" mean?

The term "control" can mean many things, such as:

- Turn the camera on or off
- Rotate the camera (turn it 360^o)
- Pan the camera (move the camera from side to side – a partial "rotation")
- Zoom the camera (increase or decrease the magnification, to interpret the picture more effectively)
- Tilt the camera (up and down)
- Display the camera's images on one or more display consoles or monitors
- Add a time stamp to a displayed camera image
- Transmit a camera image to another agency
- Record camera images for archival purposes

As used in the context of a proposed system, the term could mean all of those capabilities or merely some of them. It is the system engineer's job, in this process area, to determine which capabilities the user wants. In doing so, the system engineer may determine still more refined requirements. For example, the following requirement seems straightforward: *The TMC operator shall be able to rotate the camera 360^o in either direction (e.g., right or left) from its starting position.* But it isn't because we need to answer a few questions before we know exactly what's wanted. These questions include:

1. Once the rotation has begun, should it continue automatically or must the operator continue to direct it?
2. If the rotation continues automatically, when does it stop?
    - When it reaches its original position?
    - When the operator cancels the "rotate" command?
    - Automatically after some specified time?

3. Is there only one way to stop the camera's rotation? (E.g., by the operator canceling the "rotate" command.) If not, what options exist? If yes, what is it?
4. Must the rotation be a smooth, continuous rotation or should the camera "jump" to pre-specified points along the circle of rotation?

The above list of questions is just an illustration, to give you a feel for how a system engineer could derive additional requirements from what looks like a complete requirements statement. Note, however, that nowhere do we specify <u>how</u> the operator should "control" the camera's actions (the "command" can be issued in a number of ways, ranging from the entry of the word "rotate" as a command to pressing a button to point-and-click on a virtual button on the console display screen). We don't want to over-specify and mandate a "how" answer; we do want to make sure we've considered and included all of the functional requirements.

In addition to deriving requirements, the system engineer also decides how to allocate them to system components. Camera controls, for example, could be handled by switches, knobs, buttons, or other hardware devices located on an operator's console. Or the controls could be handled in part by hardware and in part by software. There could also be separate pieces of software for different camera functions. For example, the software to transmit a camera image to another agency could be separate from the software used to control the standard camera functions, such as pan, zoom, and tilt, since the transmission software may be used less frequently.

The requirements traceability matrix is a key project document

The system engineer allocates all requirements to the system components that will satisfy them. As part of doing this, the system engineer creates the *requirements traceability matrix*. You can use this matrix to verify that there is a system component for every requirement that needs to be met. In later stages of the project, you'll use this matrix to tell you what to test for as each system component comes on-line. The *requirements traceability matrix* is a key document (or database) for a systems project. Figure 8 shows the simplest form of a *requirements traceability matrix*, showing just which subsystems address each requirement.

Analyze Candidate Solutions. This process area deals with finding and analyzing solutions that meet the needs and goals expressed for your system. You'll perform trade-off studies in this process area and may also conduct cost-performance analyses to select the best candidate solution from those that you evaluate. Some analysis techniques include:

- Prototyping
- Simulation
- Modeling
- Trade-off study
- Decision tree
- Literature search

- Review and use of prior analyses
- Expert judgment
- Process quality improvement

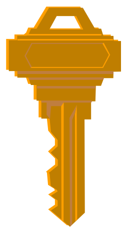
	Requirement 1	Requirement 2	Requirement 3	Requirement 4	Requirement 5	...	...	...	...	...	...	...	...	...	...	...	...	Requirement n
Subsystem 1	x			x														
Subsystem 2		x			x													
Subsystem 3			x			...	...	...	...	...	...	...	...	...	...	...	...	
...						...	...	...	...	...	...	...	...	...	...	...	...	
...						...	...	...	...	...	...	...	...	...	...	...	...	
...						...	...	...	...	...	...	...	...	...	...	...	...	
Subsystem n																		x

Figure 8
Requirements Traceability Matrix

Evolve System Architecture. The "architecture" in this process area's title is not the National ITS Architecture, although one can build a system architecture from the National ITS Architecture[12]. "Architecture" describes the overall framework of the system you are building. It refers to the physical system environment (hardware, networks, facilities) and the logical constructs (subsystems) that function in the physical system environment. As a framework, "architecture" provides a blueprint within the system design effort, giving everyone involved in the system design effort a clear understanding of what you are trying to accomplish and how. You evolve a system architecture in a top-down fashion, adding details as you flesh out the system design. There are multiple levels of architecture within a system. Each subsystem has its own architecture; each module or component of a subsystem has its own architecture. In addition, each physical item (equipment) within the system has its own architecture.

[12] The U.S. Department of Transportation (DOT) has published a rule, *23 CFR Parts 655 and 940, Intelligent Transportation System Architecture and Standards*, that defines conformance with the National ITS Architecture as requiring the development of a regional ITS architecture, i.e., a local implementation of the National ITS Architecture, within four years after the region's first ITS project advances to final design, with subsequent projects having to adhere to that regional ITS architecture. For more information, look for the architecture conformity section of the DOT's web site (www.its.dot.gov). We cover the manner in which the National ITS Architecture can support requirements development for ITS systems in a companion monograph to this one, *Developing Functional Requirements for ITS Projects*.

The other critical purpose that architecture serves is to identify all of the interfaces within the system and within each of the system's parts. The interfaces consist of the physical connections between components of their system, the logical connections between the system and other systems, and the data that flows between components or systems at each interface. You must document these details and ensure that the system, as it evolves, continues to address all of your received and derived requirements.



Architecture Establishes the System Framework and Identifies Interfaces!

Integrate System. Some base practices in this process area are obvious, others aren't. Although it may seem that integrating a system is like putting a jigsaw puzzle together, i.e., you connect the pieces where they fit as they come to you, your system's assembly may require that the pieces go together in a specific sequence. For example, when putting together a computer system, you start with the hardware. Then you add an operating system. After the operating system come any other tools, such as a database management system, on which you're going to run your applications. And so on. If it doesn't matter which comes first, the chicken or the egg, you don't worry about how you assemble your system. When the order matters, you should prepare an integration strategy and communicate it to the people who are going to integrate the pieces as you make them available.

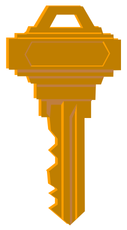
Verify and Validate System. There's an easy way to differentiate between "verify" and "validate." You "verify" when you determine that you built the system right, i.e., that the system meets the specifications that you laid down for it. You "validate" when you determine that you built the right system, i.e., that the system meets the user's needs and expectations. You verify the system by testing it against criteria you laid down to ensure it does what you intended. This is your system test plan. To validate the system, you let your users test it, using whatever criteria they have for acceptability. This is the system acceptance test plan. When you, as program manager, represent the user community and a contractor is the system developer, you approve the contractor's system test plan. However, the contractor cannot develop the system acceptance test plan. An independent group (preferably the user or with heavy user involvement) must prepare and execute the acceptance test plan.

These two tests are the basis for future end-to-end system tests you will conduct as you enhance the initial system. The details of the tests conducted may change, but the overall test plans should not.

Minimum Requirements for Systems Engineering

We've discussed the SE-CMM and its process areas. It's fairly extensive and geared to organizations that build more than one system or product. If you only expect to build one system, you just define a systems engineering process for your project. What should that process look like? James Martin believes that the following are the minimum requirements for an effective systems engineering process:

"In general, every project ought to have a schedule (produced on one of the popular project management programs such as TimeLine, or Microsoft Project), a requirements traceability matrix which includes tests (for small enough projects use a commercial database such as dBase, Paradox, or Access), configuration management, and a modification request system.



Document All of Your Decisions and The Reasons for Them!

Documents produced (if not specified by the contract) should include as a minimum the plan itself, the schedule (including resources, costs, and dependencies), a requirements specification, the requirements traceability matrix (which can include system tests), a configuration item list for hardware and software in sufficient detail that all personnel know what they are producing, and a set of modification requests which undergo periodic review."[13]

What it boils down to is this. You must have a plan and a schedule for your project. The difference between the two is that the plan describes what needs to be done and the schedule shows when you want it done, by whom, and how long you expect it to take. The schedule also shows the order in which tasks should be done and which ones have to wait until others are completed before starting. You also must have requirements written down and know what parts of your system you're going to use to meet each one. You must know all of the parts in your system and track each part and its status. You must describe each part in sufficient detail that the people who are building it can build it correctly. And you need a procedure that allows people to recommend changes to any part of the system, to assess the value of the change, and to decide whether to accept the change. Once you agree to change the part, you must communicate the changes to everybody who is making or using the part, so they know how it changed and why. You should write down every decision that you make along with the reasons that you made that decision. Like real estate, systems engineering has three key things: "Document, document, document."

The next section looks at different ways of dealing with risk and uncertainty in the systems engineering process.

[13] Martin, James N. *Systems Engineering Guidebook: A Process for Developing Systems and Products (Systems Engineering Series),* CRC Press, Boca Raton: 1996, pp.183-184

4.0 Addressing Uncertainty and Risk With Systems Engineering

Types of Uncertainty and Risk Addressed

One of our major premises is that reducing uncertainty and risk in projects through sound systems engineering techniques significantly improves the project's chances for success. Some of the factors that cause uncertainty and risk in projects include:

- Complexity
- Processing capacity/communication bandwidth
- Rate of technological change
- Information imperfections

Let's consider how each of these affects uncertainty and risk.

Complexity. The more parts that a system has, the more complex it is likely to be. When a system includes software that does more than just trivial processing, the software alone brings considerable complexity to the system. Complexity in a system adds uncertainty in two ways. First, it is difficult to visualize a system when there are multiple interactions among all of its parts. Since we have difficulty "seeing" it, we may have difficulty in understanding and describing it. When we can't create a comprehensive picture of our system, whether with pictures or words, we may omit some relevant part of it. This, in turn, means that we may not fully satisfy all of the requirements that our users had in mind. Second, we may find it difficult to maintain the interfaces and interactions within the system whenever we change any of its parts. Since some part of the system will change either while we're building it or after we field it, change in a complex system can have unexpected (and undesirable) results.

Processing Capacity/Communication Bandwidth. When we build a system, we may be uncertain about the capacity required to process the data that we want it to handle. This uncertainty can lead to one of three possible outcomes. First, we estimate the processing capacity correctly and the system handles our data volume satisfactorily. Second, we overestimate the processing capacity required and our system has excess capacity. Third, we underestimate the system capacity and the system cannot handle the actual volume. The second case is a problem if the cost of acquiring the excess capacity causes us to skimp on some other part of the system. The last case is clearly a problem.

Underestimating system capacity is a common problem with ITS projects, particularly with regard to the computing and telecommunications needs of the project. In large part, this is because ITS project managers tend to be transportation engineers and thus not very familiar with software or telecommunications systems. To avoid having this be one of your problems, you should include computing and telecommunications experts as "specialty" engineering capability.

Processing capacity is not just a matter of computer hardware. We may design processes to be done by humans and not consider the information overload that they may have to deal with. For example, consider a process where humans received data from field locations and have to update data records to keep the information in a system current. If the receipt of field data is spaced so that a human operator has enough time to record all of the data and ensure that the recording is accurate, processing capacity is adequate. However, if the data is received so rapidly that operators cannot keep up with it, transcription errors will occur and the quality of the database will suffer. Designing human interfaces is not just a process of making screens attractive to users and having icons to identify common functions. It also involves ensuring that users can do their jobs in the time allotted for them.

Rate of Technological Change. The rate of technological change in today's environment, particularly in computer technology and in telecommunications, is very rapid. For example, the processing power and speed of computer chips has doubled approximately every 18 months. And computers are not the only electronic devices whose evolutionary pace is very rapid. Since they were first introduced in the early '80s, cellular phones have shrunk in size, weight, and price while increasing in capability. The Internet also has shown explosive growth in the last 15 years. With the rapid advances in technology and related standards, technical solutions that are state-of-the-art today may be obsolete in a few short months; solutions that are technically infeasible today may be practical in just a few months more. Given the rapid change in technical capability, projects face uncertainty in choosing the right technology. Yet, project managers have to make choices about what technology to deploy if they are going to make schedules and implement systems. Waiting for the next great advance is rarely an option.

Information Imperfections. Information imperfections are those deficiencies in what we know that make it difficult to make a choice with which we are comfortable. The imperfections can result from either the quantity of information or from its quality. If we don't have enough information, our uncertainty stems from not knowing all of the facts we need to make the best choice. If we have too much information, we have to sift out the relevant from the clutter. If we have faulty information, we will make bad decisions believing them to be right. Our goal in systems engineering is to determine whether we have information imperfections and to improve the information that we have.

Decision-Making Under Uncertainty.

Managers are generally averse to making decisions when there is risk about the decision. Unfortunately, in real life, managers have to make many decisions with imperfect information and high degrees of uncertainty. As a systems engineer, you must help managers by giving them ways to assess the risks involved in a decisions and the strategies they can follow to reduce the risk.

The worse possible case for decision-making is the case of complete uncertainty. There are several strategies available to a manager in this situation, but all require you to be able to determine the possible results of your actions. Let's take the case where a manager is attempting to choose between two alternatives with each alternative having two possible results. This gives us a 2x2 outcome space, illustrated in Table 5. Let's assume that the first alternative, Alternative 1, has two possible outcomes. The first is an outcome with a positive value of $2,500; the second is a negative outcome, losing $500. We represent these two outcomes in the first row of Table 5, with the negative outcome, a loss, shown as -$500. Alternative 2, on the other hand, has the same positive value of $500 for each of its two outcomes. The second row in Table 5 illustrates this alternative.

Table 5
Alternative Assessment Matrix

Alternative	**Result 1 Value**	**Result 2 Value**
Alternative 1	$2,500	-$500
Alternative 2	$ 500	$500

If these are the possible outcomes, how should we choose an alternative?

Games theory offers three alternative strategies:

- Maximin: In this strategy, you choose the alternative that optimizes the minimum result. This is the same as choosing the alternative with the least harmful worse case scenario. Given the above potential results, you would choose alternative 2.

- Maximax: In this strategy, you choose the alternative that optimizes the maximum result. This is choosing the alternative with the best upside. In the case illustrated by table 4, you would choose alternative 1.

- Equal probability rule: In this strategy, you assume that all alternatives have the same probability of occurring and choose the one that yields the higher *expected value*. Expected value is determined by multiplying the value of each result by the probability of its occurring. Since we have two results for each alternative, each has a probability of 50%. The expected value of alternative 1 is 1,000 ((2,500 * .5) + (-500 *.5)) and the expected value of alternative 2 is 500. This strategy would pick alternative 1.

Two of the three strategies opt for alternative 1 while only one recommends alternative 2. So which would a manager choose?

In all likelihood, a manager will use the maximin strategy and choose alternative 2. Why? Because it offers him or her the least risk if he/she is wrong. Either result in alternative 2 guarantees a positive return. Even if alternative 1 offers a 50% probability that the result may be a greater return than either result in alternative 2, it is the 50% probability of the negative result that dissuades many managers. The *psychological* difference in the two alternatives favors the second.

Our goal in systems engineering is to improve the quality of the information that we can provide project managers on ITS projects. By doing so, we take the decisions they have to make out of the realm of complete uncertainty and give them a better chance of making high quality decisions.

Let's look at some of the tools available to us.

Systems Engineering Tools for Addressing Uncertainty and Risk

There are seven such tools that we'll consider:

- Project scheduling and tracking tools
- Trade-off studies
- Reviews and audits
- Modeling and simulation
- Prototyping
- Benchmarks
- Technical performance measures

In our discussion, we'll talk about how these tools address the issues we described.

Project Scheduling and Tracking Tools.

There are three basic scheduling and tracking tools:

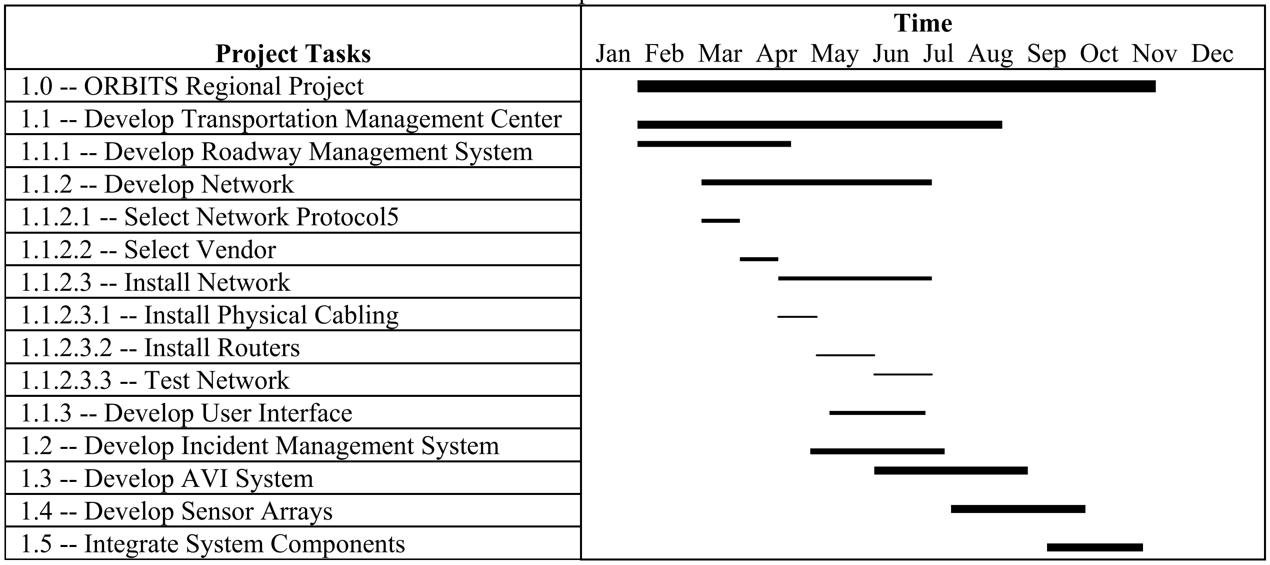
- Gantt charts
- Activity networks
- Work breakdown structures

Most of the automated tools available today allow you to create and represent your project schedule in each of these formats, transforming from one format to another automatically upon your request. Each tool (format) has a particular strength.

> **Gantt Charts.** The Gantt chart, or "bar chart," is a common way of depicting the tasks on a project against their expected duration. Figure 9 is a sample Gantt chart.
>
> This figure uses the same tasks as those shown in the sample Work Breakdown Structure figure (Figure 7) in section 3.

The strength of a Gantt chart is that it shows the duration of each task, along with the expected start and end dates, explicitly. It may or may not show the dependencies among the tasks, depending on what you use to produce it.

Figure 9
Sample Gantt Chart



Activity Networks. An activity network shows tasks and the order in which they should execute. There are different ways to draw activity networks, but we'll illustrate one using the "activity on arrow" approach, which lists the tasks on the arrows connecting the nodes of a network. Figure 10 illustrates this type of network.

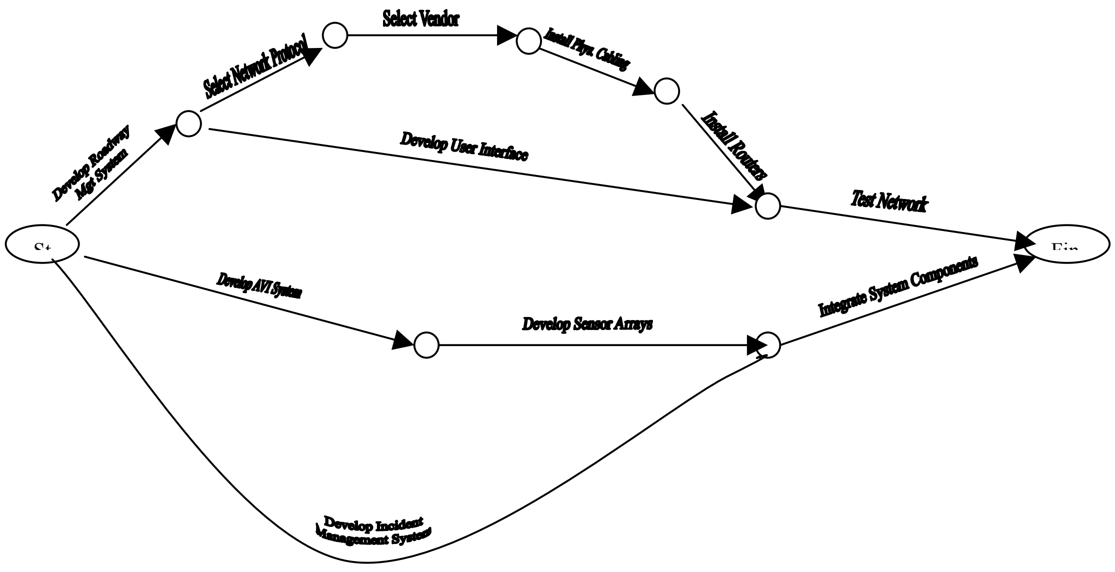


Figure 10
Activity Diagram

The strength of an activity network is that it clearly shows the sequence of tasks and the dependency of any task on another. It frequently is used to show the *critical path* of a project. By definition, the *critical path* is the sequence of tasks in which a delay in any task causes an overall delay in the completion of the project. A project manager must monitor tasks on the critical path more closely than tasks that are not. However, the critical path is not constant. If some tasks not on the critical path are delayed beyond a reasonable point, they may change the project's critical path and cause an unexpected delay. Thus, a project manager has to monitor all tasks in a project schedule and continually verify that the critical path has not changed.

Work Breakdown Structure. The third type of scheduling and tracking tool is the Work Breakdown Structure (WBS). We discussed this briefly in section 3. In a Work Breakdown Structure approach, a project manager divides the project into "work packages," or tasks that can be grouped together to yield a distinct product on the project. On an ITS project, for example, a work product may equate to a market package in the National ITS Architecture. The effort required to implement one of these market packages could be part of a work package. The work package can be broken down into smaller and smaller work packages until we reach the level where work is being performed by one individual on the project team with a definite, measurable product at the end of the effort.

The strength of a WBS approach is that it allows you to show the hierarchy of work clearly. Lower level tasks combine to form higher level tasks. You can easily assign resources to a task and calculate what the resources will cost you. Then, by adding up the cost of all of the lowest level tasks, you can get the total cost of a work package or of the entire system.

These techniques illustrate how one can deal with the complexity in a system. By dividing the work required to implement the system into small manageable pieces, regardless of the technique used, and then estimating the effort to produce each piece, a project manager can create a schedule and track progress against it. Moreover, each member of the project team can understand his or her role and responsibilities on the project and focus on the work he or she must accomplish to make the project a success. These tools, while seemingly mundane, are valuable elements in the systems engineer's toolbox.

Trade-off Studies.

Trade-off studies are one way to deal with information imperfections and with rapid technological change. While the trade-off study is sometimes used to decide whether a technical solution to a problem exists, a trade-off study can examine different technical alternative solutions to a problem. Trade-off studies can be paper studies where the analyst assesses different products or solutions using technical descriptions from the vendor of the product. In this case, the analyst is assessing how the technical description of the product maps against the requirements the project has. A paper study, however, relies on someone else's assessment of the technical capability of a product. If that assessment is wrong or incomplete, your analysis will suffer. Another approach to conducting a trade-off study, although this option is not always possible, is to test competing products in a controlled environment that simulates, as well as possible, the actual conditions under which the products will be used. This approach can yield the most reliable results, if the test environment is realistic.

Determining the availability of a solution or assessing the different technical solutions to a problem are ways to deal with information imperfections. Trade-off studies can also be used to address technological change. These are more predictive studies that look at trends in technology and even at what technologies are in the research laboratories at major manufacturers. The studies are attempts to forecast when advances in a technical area are likely to move from the laboratory to the marketplace. This type of study is useful if it is possible to postpone a decision about the technology to be used on a project until a point closer to the time when you want to implement the system. They are also useful if you want to build flexibility into the system so that you can more easily take advantage of advances in technology.

Reviews and Audits.

Design reviews help ensure that the system design, as it evolves, continues to meet the needs of the end-user and all others who interact with it. The basic goals of a design review are:

- Ensure that the evolving system design meets the requirements defined for the system
- Ensure that the evolving system design is understood and agreed to by the stakeholders affected by the system
- Find any flaws in the design, so that they can be corrected before they become more firmly embedded in the system implementation
- Communicate the evolving design to all interested parties

In section 3 of this paper we specifically called out three kinds of reviews: System Requirements Review, Critical Design Review, and Preliminary Design Review. The Preliminary Design Review and the Critical Design Review are the two key types of reviews that should be conducted on ITS project design elements.

There are other types of reviews and audits that you can conduct as part of the system engineering toolbox. You can find them described in the References to this document. We will not go into more detail in this area.

Modeling and Simulation.

One way to examine potential bottlenecks and impediments to system performance is to model the system and simulate its operation under different conditions of load. This is particularly useful if the system has not yet been built and you want to check out possible performance problems before you build it. Modeling will help identify potential choke points (parts of the system where capacity limitations prevent the system from processing the expected volume it must handle). It can also help identify the conditions of load under which the system ceases to perform adequately. Once you have identified these potential problem areas, you can modify the system design to eliminate them or to reduce their negative impact on the system. Modeling and simulation are ways to reduce uncertainty about system performance. To a degree, they also help deal with issues of information imperfection when you want to examine the system performance under different load scenarios. Modeling and simulation can then help determine the ranges of system performance and the ranges of volume that the system can handle before becoming saturated.

Let's consider how we might use modeling and simulation in an ITS project. Let's say that Chief Traffic Engineer Mandrake is considering a project to give traffic signal priority treatment to buses in the Metropolis metropolitan area. (This capability already exists for the Metropolis emergency vehicles and it can be expanded to the transit vehicles.) Giving Metropolis Transit Authority buses traffic signal priority treatment will help them keep to their schedules and provide better overall service. However, some of Mandrake's traffic engineers are concerned about the effect on traffic on cross-streets. Mandrake agrees that this could cause problems and is only interested in a selective priority approach, giving priority only to those buses running behind schedule.

To examine the potential congestion problems that could arise, Mandrake commissions a simulation study. She asks her systems engineers to model the bus routes with the greatest number of behind-schedule trips over the last year, looking at the potential effect on schedule performance if buses on those routes get signal priority treatment when they're running behind schedule. She wants to know the answers to questions like the following:

- Will more buses be on schedule if they're granted traffic signal priority when they fall behind schedule?
- How much traffic build-up will occur on cross-streets?
- Which cross-streets will have the worst build-ups?
- At what times will the worst traffic build-ups most likely occur?

Mandrake assigns some of her staff to collect the necessary data and then to give it to the modelers among her systems engineers. She tells the modelers what results she wants and lets them analyze the data and prepare their simulations to answer her questions.

When the modelers finish their work, they'll present Mandrake with the information she can use to make her decision. If the selective priority scheme isn't effective, the modelers can show Mandrake why. If it is, but causes congestion on some cross-streets, they can tell Mandrake where and when the congestion will most likely occur. Mandrake may still proceed to implement this priority system, but will do so knowing the consequences of her action.

Prototyping.

ITS projects have used prototyping in many ways. One key use of prototyping is to develop system requirements for complex ITS systems. The State of Maryland recently used prototyping creatively to determine its system requirements for a major ITS effort. The State hired two vendors to build a prototype of the ITS system that Maryland wanted to implement. Each vendor had to dig out the critical requirements from the user community and had to develop approaches to solve the technical problems that arose in building their prototypes. Maryland was then able to compare the two prototypes and select the one that they believed had the better chance of providing an acceptable solution. The winning vendor had already invested considerable effort in building the prototype and was well positioned to complete a successful build of the overall system.

Prototyping can also be used to create a version of the system that can be used for benchmarking system performance. The prototype is an operational model of the system, but in a scaled down format. Prototyping is also valuable in determining whether it is possible to interface elements of a system that have never been linked together before. This helps reduce the uncertainty related to the interfaces.

Benchmarks.

Benchmarks are useful when you want to measure the actual performance of a system and ensure that it can meet its performance requirements. Benchmarks can be conducted on the implemented version of a system, as part of the system or acceptance testing of the system. They can also be conducted against a scaled-down version of the system to assess how well the scaled-down version performs, before investing in the final version of the system. When you use a scaled-down version of the system in a benchmark, you must be confident that increasing the scale of the system will not significantly change system performance. Otherwise, the benchmark may be invalid. Benchmarking is particularly useful in measuring the capability of telecommunications systems and the process capability of computing systems.

Technical Performance Measures.

Technical performance measures are those characteristics of a system that directly impact its performance. For example, the weight of an airplane directly affects its speed and cargo or passenger capacity. In an ITS setting, CPU utilization and processing time are two technical performance measures to consider. Storage capacity, i.e., the amount of storage necessary to hold the data required for normal system operation, is another possible technical performance measure to use. Other examples could include: number of sensors supported in simultaneous operation; total time (e.g., .5 seconds) to process toll collection information from an AVI tag; end-to-end time for signal light control processing.

Technical performance measures should be established early on in the ITS project. Then, as the system develops, the actual system values achieved for each of these technical performance measures is compared against its design goal. For example, if you defined toll collection processing time goal as .5 seconds, measuring actual processing time at .75 seconds indicates that you exceeded the technical performance measure by 50%, an undesirable result. You would need to modify the design or its implementation to reduce response time and achieve the defined goal.

Having discussed some system engineering tools, let’s look at what other sources of help exist.

5.0 Where to Go to Get More Help

Department of Transportation Resources

One Department of Transportation resource is the ITS System Acquisition document set, *The Road to Successful ITS Software Acquisition, Volumes 1 and 2,* and its related course. A second document resource, one that covers basic system engineering concepts in the context of Transportation Management Centers (TMCs) is the *Transportation Management Center Concepts of Operation Implementation Guide*. Other documents that can provide useful insights into the systems engineering effort are the companion documents to this one: *Developing Functional Requirements for ITS Projects*, and *The ITS Guide to Configuration Management*.

The National ITS Architecture is another resource that can help you with your systems engineering needs as it provides an excellent starting point for defining requirements (and for developing regional ITS architectures). It also provides guidance, through its market packages, on implementation strategy for ITS systems.

National Highway Institute

The National Highway Institute is offering two introductory Systems Engineering courses for transportation engineers and has other, more detailed courses that cover topics related to systems engineering. Contact the NHI for information on the course numbers and course schedules.

Standards

There are two existing overview or general standards for systems engineering: the Electronics Industry Association (EIA) and American National Standards Institute (ANSI) joint standard, EIA/ANSI 632, and the Institute for Electronic and Electrical Engineering (IEEE) standard, IEEE 1220. These two standards should be joined by a third, published by the International Organization for Standardization (ISO), ISO 15228. The ISO standard is still in the Draft International Standard (DIS) stage as of August 2001. All three standards have a common heritage and are similar, but not identical. Any of the standards could be used in an ITS project.

In addition to the general or overview standards, there are a number of software engineering, configuration management, and quality standards that you can use. Table 6 provides a representative sample of these standards. You can search for other relevant standards from the following organizations (with web sites shown in parentheses):

- AIAA American Institute of Aeronautics and Astronautics (www.aiaa.org)
- ANSI American National Standards Institute (www.ansi.org)
- EIA Electronics Industry Alliance (www.eia.org)
- IEEE Institute of Electrical and Electronics Engineers (www.ieee.org)

- ISO International Organization for Standardization (www.iso.ch)

Table 6
List of Representative Standards

Standard	Title of Standard
AIAA R-013	Recommended practice for software reliability
AIAA G-043	Guide for the preparation of operations concept documents
ANSI/EIA 632-98	Processes for engineering a system
ANSI/EIA 649-98	National consensus standard for configuration management
IEEE Std 730 1998	Standard for software quality assurance plans
IEEE Std 730.1 1995	Guide to software quality assurance planning
IEEE Std 828 1998	Standard for software configuration management plans
IEEE Std 829 1998	Standard for software test documentation
IEEE Std 830 1998	Recommended practice for software requirements specifications
IEEE Std 1008 1987	Standard for software unit testing
IEEE Std 1012 1998	Standard for software verification and validation
IEEE Std 1016 1998	Recommended practice for software design descriptions
IEEE Std 1016.1 1993	Guide to software design descriptions
IEEE Std 1028 1997	Standard for software reviews and audits
IEEE Std 1059 1993	Guide for software verification and validation plans
IEEE Std 1062 1998	Recommended practice for software acquisition
IEEE Std 1063 1987	Standard for software user documentation
IEEE Std 1058 1998	Standard for software project management plans
IEEE Std 1219 1998	Standard for software maintenance
IEEE Std 1228 1994	Standard for software safety plans
IEEE Std 1233 1998	Guide for developing system requirements specifications
ANSI/IEEE Std 1042	Guide to software configuration management
IEEE Std 1348 1995	Recommended practice for the adoption of CASE tools
ISO/IEC 9126 1991	Software product evaluation -- Quality characteristics and guidelines for their use
IEEE/EIA 12207.0 1996	Software life cycle processes
ISO/IEC 12207 1995	Software life cycle processes
IEEE Std 1465 1998	Software packages: -- Quality requirements and testing (Adoption of ISO/IEC 12119 1994
IEEE Std 1489 1999	Standards for data dictionaries for Intelligent Transportation Systems
IEEE Std 1362	Guide for concept of operations document
ISO/IEC 12119 1994	Software packages -- Quality requirements and testing

References

The texts listed in the bibliography to this paper are all useful for systems engineering. The *Handbook of Systems Engineering*, while the most expensive, is also the most comprehensive. Also, INCOSE (see below) publishes its *Systems Engineering Handbook*, subtitled *A "HOW TO" Guide For All Engineers*. This is a very practical handbook and would be a useful addition to any engineer's library.

Other

There are a number of systems engineering organizations world wide, all of which can provide help in applying systems engineering practices on an ITS project. The International Council on Systems Engineering (INCOSE), located in the United States, is recognized as the leading systems engineering organization in the world. Its web site (www.incose.org) provides links to a number of other systems engineering resources. Other systems engineering organizations include:

- System Design Engineering, University of Waterloo, Canada
- WG Systems Engineering, University of Munich, Germany
- Norwegian Systems Engineering Council (NORSEC)
- Systems Engineering Society of Australia (SESA)
- Defense Evaluation and Research Agency (DERA) of the United Kingdom

Also, Southern Methodist University (SMU), in Texas, has 1000+ links to systems engineering resource sites on a web page, (www.seas.smu.edu/disted/sys/r.html).

Appendix
Earned Value Analysis Calculations

Earned Value Analysis Calculations

Calculating Earned Value on ITS Projects

In analyzing a project's cost performance in terms of its schedule, it is not enough to compare actual costs incurred (as a percentage) to project time elapsed (as a percentage). A project could be 75% through its planned life, have spent 75% of its planned costs, yet only have done 50% of the work required. If this is true, the project is not performing well. Earned value analysis is a method for estimating cost and schedule deviations during a project and projecting those deviations to the end of the project. In conducting an earned value analysis, you look at three cumulative costs on the project:

- Budgeted cost of work scheduled (BCWS)
- Budgeted cost of work performed (BCWP)
- Actual cost of work performed (ACWP)

With these three values, you can estimate four key factors:

- ***Cost variance (CV)***: This is the difference between the budgeted cost of work performed and the actual cost of work performed, using the equation:

  $$CV = BCWP - ACWP$$

- ***Schedule variance (SV)***: This is the difference between the budgeted cost of work performed and the budgeted cost of work scheduled, using the equation:

  $$SV = BCWP - BCWS$$

- ***Estimated cost at completion (ECAC)***: This is a linear extrapolation, using the original cost budget (CB) and the ratio of actual cost of work completed versus budgeted cost of work completed. The equation is:

  $$ECAC = (ACWP/BCWP) \times CB$$

- ***Estimated time to complete (ETC)***: This is also a linear extrapolation, using the original schedule, in time (T), and the ratio of budgeted cost of work scheduled versus budgeted cost of work completed. The equation is:

  $$ECAC = (BCWS/BCWP) \times T$$

Let's take a numeric example to demonstrate how this works.

We have a project whose original schedule was one year. The original cost estimate for the project was $500,000. We are now reviewing the project at the end of six months, and we determine the following facts:

- Our project has been spending about $40,000 per month and total expenditures for the first six months are $235,000.

- The project has six major work breakdown schedule (WBS) tasks at the second detail level, and three of them are complete.

- The tasks that are complete are tasks 1.1, 1.3, and 1.4.

- The estimated costs of each major WBS task are:

  - Task 1.1 -- $ 75,000
  - Task 1.2 -- $ 90,000
  - Task 1.3 -- $ 75,000
  - Task 1.4 -- $ 65,000
  - Task 1.5 -- $100,000
  - Task 1.6 -- $ 95,000

- At the end of six months, you planned to have tasks 1.1, 1.2, and 1.3 completed.

That's enough detail for our simple example.

Let's determine each of the values we need. We already know the actual cost of work completed ($235,000). We can compute the other two costs.

Budgeted cost of work performed is $75,000 + $75,000 + $65,000 or $215,000. Budgeted cost of work scheduled is $75,000 + $90,000 + $75,000 or $240,000. With these three values, the total budget, and the total time estimate we can calculate our project deviations.

CV = $215,000 - $235,000 or -$20,000. That means we have a negative cost deviation. This is not good.

SV = $215,000 - $240,000 or - $25,000. We also have a negative schedule deviation. More bad news.

When we calculate the estimated cost at completion, we see the potential impact of the cost variance: $500,000 x ($235,000/$215,000) = $546,500 or a 9.3% potential cost overrun. And calculating the estimated time to complete tells us what kind of extra time we may need: 12 months x ($240,000/$215,000) = 13.4 months or an 11.7% time overrun.

Well, the news is not good anywhere. But remember, this is a management tool. If you know now that there's a potential to overrun both your costs and your schedule, you still have time to take corrective action. If you can't fix the problem that's causing your project to overrun, at least you can alert your management ahead of time.

Granted, the example given is very simple. And, you could ask, why did we ignore the work completed in the other tasks, which could have changed the calculations dramatically? In reality, you're going to calculate the actual cost and the budgeted costs at the lowest level of the WBS, the level where the task is either done or not done. (You should only consider a task "x percent complete" when there is a solid measure that tells you that the "x" represents a value that you can count on.) You can adjust how you credit work done and work scheduled, but remember that you're trying to get a realistic picture of where you stand on your project, not make yourself look good when the circumstances don't warrant it.

So good luck! This may provide one more good tool in your systems engineering tool bag.

Bibliography

______, "Payroll Systems Flunk Final Exam," *Computerworld*, October 25, 1999

Anon., *Systems Engineering Handbook, A "HOW TO" Guide For All Engineers, Release 1.0*, International Council on Systems Engineering (INCOSE), San Francisco Bay Area Chapter: 1998

Blanchard, Benjamin S. and Wolter Fabrycky, *Systems Engineering and Analysis, 3rd ed.*, Prentice Hall, New Jersey: 1998

Blanchard, Benjamin S., *Systems Engineering Management, 2nd ed.*, John Wiley & Sons, Inc., New York: 1998

Boehm, Barry W., (ed.), *Software Risk Management,* IEEE Computer Society Press, Washington, DC: 1989

Charette, Robert N., *Software Engineering Risk Analysis and Management*, McGraw-Hill Book Company, New York, 1989

Eisner, Howard, *Essentials of Project and Systems Engineering Management*, John Wiley & Sons, Inc., New York: 1997

Hunger, Jack W., *Engineering the System Solution: A Practical Guide to Developing Systems*, Prentice Hall, Inc., Englewood Cliffs, NJ: 1995

IEEE Std. 1362-1998: *IEEE Guide for Information Technology – System Definition – Concept of Operations (ConOps) Document*, Institute of Electronic and Electrical Engineers, 31 December 1998

Janis, Irving L. and Leon Mann, *Decision Making: Psychological Analysis of Conflict, Choice, and Commitment*, The Free Press, New York: 1977

Johnson, Jim, "Turning Chaos Into Success," *Software Magazine*, Dec. 1999 – Jan. 2000

Martin, James N. *Systems Engineering Guidebook: A Process for Developing Systems and Products (Systems Engineering Series),* CRC Press, Boca Raton: 1996

McQueen, Bob and Judy McQueen, *Intelligent Transportation Systems Architectures,* Artech House, Inc., Norwood, MA: 1999

Ould, Martyn A., *Strategies for Software Engineering: The Management of Risk and Quality*, John Wiley & Sons, Inc., New York: 1990

Rechtin, Eberhardt and Mark W. Maier, *The Art of Systems Architecting (Systems Engineering Series)* CRC Press, Boca Raton: 1997

Sage, Andrew P., *Systems Engineering,* John Wiley & Sons, Inc., New York: 1992

Sage, Andrew P. and William B. Rouse, (ed.), *Handbook of Systems Engineering*, John Wiley & Sons, Inc., New York: 1999

Thome, Bernard, (ed.), *Systems Engineering: Principles and Practice of Computer-Based Systems Engineering*, John Wiley & Sons, Ltd., Chichester: 1993