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[1]. 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.

  > 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.

[1] 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.

- ***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[2] 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.

> **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

---

362-1998, *IEEE Guide for Information Technology – System Definition – Concept of Operations ocument*

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 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.

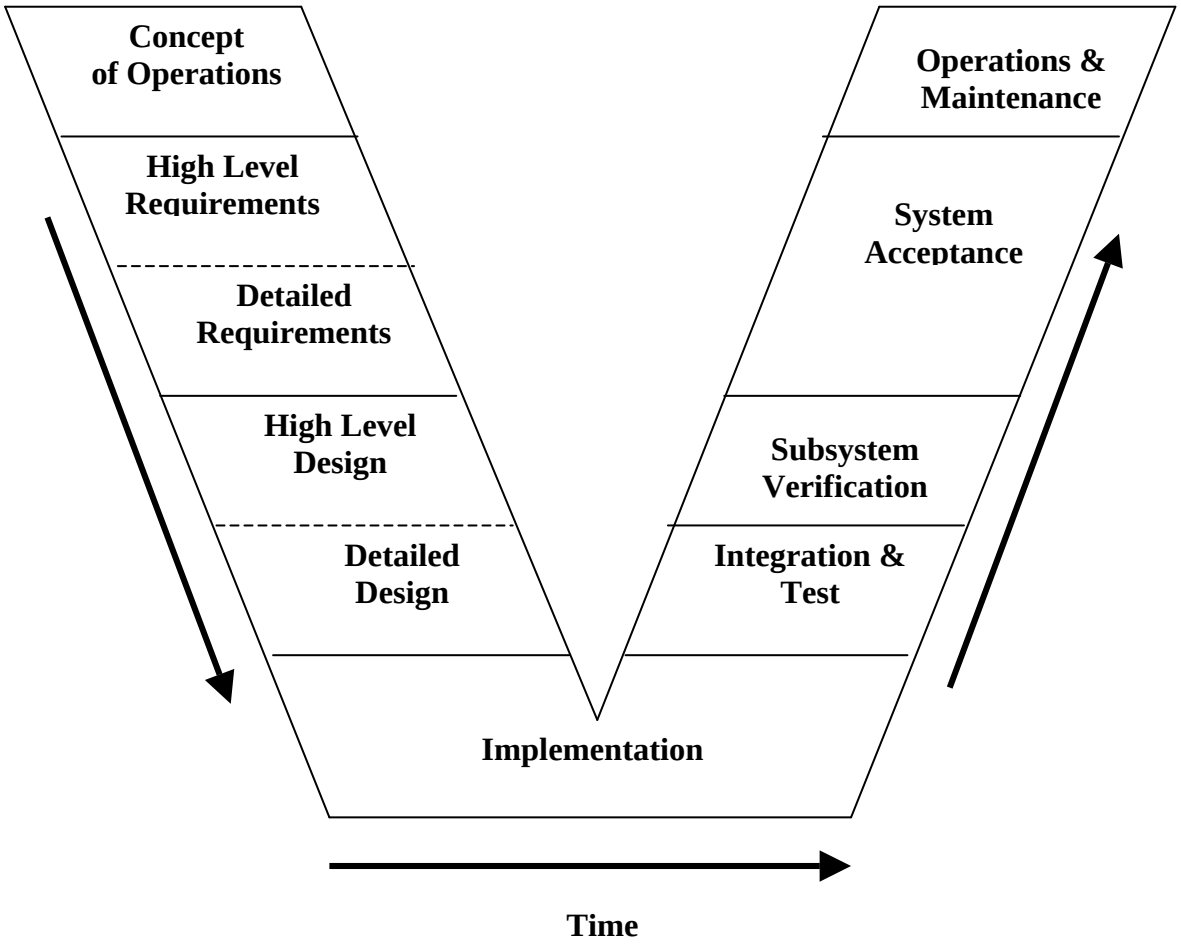

Figure 6
"V" Diagram

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 how 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.