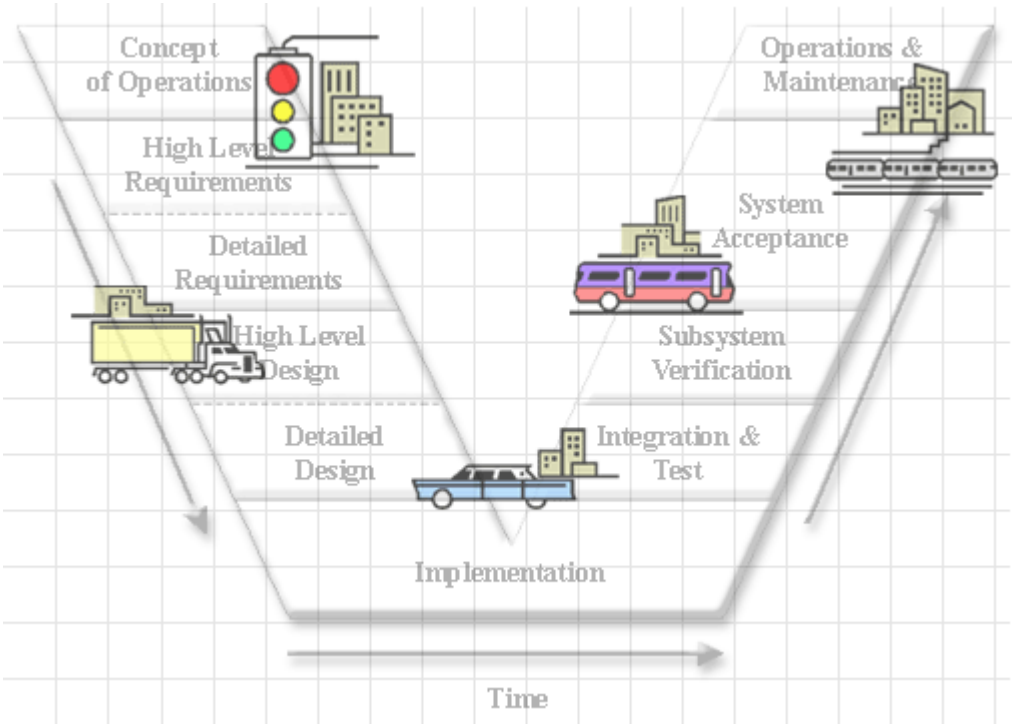
A Guide to Configuration Management for Intelligent Transportation Systems

April 2002



Prepared for

Intelligent Transportation Systems
Joint Program Office
US Department of Transportation

By Mitretek Systems, Inc.

FHWA-OP-02-048
EDL #13622

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FHWA-OP-02-048		

4. Title and Subtitle	5. Report Date
A Guide to Configuration Management for Intelligent Transportation Systems	April 2002
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Paul J. Gonzalez	

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
Mitretek Systems, Inc. 600 Maryland Avenue, SW, Suite 755 Washington, DC 20024	11. Contract or Grant No. DTFH61-00-C-00001

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
Department of Transportation Intelligent Transportation Systems Joint Program Office 400 Seventh Street, SW – Room 3416 Washington, DC 20590	14. Sponsoring Agency Code HOIT

15. Supplementary Notes
William S. Jones – Task Manager

16. Abstract

This monograph is one of a series 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 covers the topic of Configuration Management. Configuration Management (CM) formalizes the process of making changes to a system under development so that the system's builders maintain an appropriate configuration record and can always ensure that they know what the correct version of the system consists of. It is intended to establish and maintain the consistency of a system's performance, functional, and physical attributes with its requirements, design, and operational information throughout the system's life and is considered a "best practice" within the system engineering discipline.

Managing changes to requirements is essential to minimizing cost and schedule overruns on transportation system projects. CM's ability to control changes to requirements is a major reason for employing it on ITS projects.

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

17. Key Word	18. Distribution Statement
Systems Engineering, Intelligent Transportation Systems, ITS, Transportation System Projects, Configuration Management, CM	No Restrictions This document is available to the public.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	51	NA

Table of Contents

List of Figures

Figures

Executive Summary

Purpose of This Document

This document is one in a series of monographs on systems engineering topics developed to help introduce this discipline to ITS project managers and their staff. While none of these documents will turn the reader into an experienced practitioner in the area of systems engineering, they are intended to give their audience a sense of how the topic covered applies to Intelligent Transportation Systems (ITS). This monograph covers the topic of *Configuration Management*. Although other monographs in this series discuss configuration management briefly, none goes into the topic at the same level as this one.

Intended Audience

Our intended audience includes:

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

Our primary focus is on ITS project managers, but we hope to encourage others to learn more about this area. Configuration management is a key applied systems engineering practice, used to get systems built with maintainability and long-term health.

Origins of Configuration Management

Configuration management, as a systems engineering discipline, began in the 1950s, when the U.S. began developing tactical and strategic ballistic missiles. U.S. manufacturers found that they had difficulty in mass-producing missiles from their successful prototypes because they'd failed to record the contents of the successful configuration in an organized manner. ANA (Army, Navy, and Air Force) Bulletin 390 was the first configuration management guide. It formalized the process of making changes to a system under development so that the builders maintained an appropriate configuration record. Thus, once the DOD accepted a prototype for production use, the manufacturer could mass-produce the weapon. Over time, configuration management has become an accepted practice in the manufacturing and software development industries.

Current Configuration Management Standards

The two key broad configuration management standards are International Organization for Standardization (ISO) ISO 10007:1995, *Quality management – Guidelines for configuration management* and the American National Standard Institute (ANSI) and

Electronic Industries Alliance (EIA) joint standard ANSI/EIA 649-1998, *National Consensus Standard for Configuration Management*. Both of these documents are equally applicable to hardware or software configuration management. The Institute of Electrical and Electronics Engineers (IEEE) also has a standard relating to configuration management, IEEE Std 828-1998, *IEEE Standard for Software Configuration Management Plans*, which is part of the IEEE's software engineering standards.

The two general standards for configuration management provide guidance on how to employ configuration management on a project. The IEEE's software engineering standard provides more specific guidance on what to do for software configuration management. A public sector project manager for an ITS project should look at the IEEE standard to see what a software development contractor should have in a software configuration management plan for a project.

Configuration Management Principles

Configuration management is defined as: "A management process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design and operational information throughout its life."[1] Figure ES-1 illustrates how configuration management fits into the overall systems engineering process. What this figure shows is that configuration management pervades the systems engineering process, since it is a "best practice." Table ES-1 is a checklist, taken from material[2] published by the Software Program Managers Network highlighting points that they believe are part of implementing configuration management as a "best practice."

System Life Cycle Stage	Config Mgmt Planning	Config Identification	Config. Control	Config. Status Accounting	Config. Audits
Conception	√				
Requirements Analysis		√	√	√	
Design		√	√	√	
Implementation		√	√	√	
Integration & Testing			√	√	√
System Acceptance			√	√	√
Operation and Maintenance			√	√	

Figure ES-1
Configuration Management in the System Life Cycle Stages

Configuration management involves five major areas:

[1] ANSI/EIA 649-1998, *National Consensus Standard for Configuration Management*
[2] Taken from Software Program Managers Network, *The Condensed Guide to Software Acquisition Best Practices*, p. 18, and *Little Yellow Book of Software Management Questions*, p.14

- **Configuration management planning** – involves making decisions about what needs to be controlled within a product configuration, when you establish a controlled configuration, how you change a controlled configuration, and what amount of effort you're going to expend to manage configurations, with the decisions formalized in a *configuration management plan*.

Table ES-1
Configuration Management Checklist

√	Is there a documented configuration management process for this project?
√	Is the configuration management process integrated with the project plan and an integral part of the culture?
√	What classes of information does your project control?
√	What items are under control?
√	How is the decision to control them made?
√	Are all versions controlled?
√	Are configuration control tools used for status accounting and configuration identification tracking?
√	Are periodic reviews and audits in place to assess the effectiveness of the configuration management process?
√	Are all pieces of information shared by two or more organizations placed under configuration management?
√	Who on the project is responsible for change control of baselined and non-baselined items?
√	Do you have a configuration control board? If so, who are its members?
√	Do you have a process for controlling non-product software that is shared?
√	How does the developer make releases to the acquirer?
√	How does the acquirer take delivery of items from the developer?

- **Configuration identification** – involves identifying the *configuration items* and the unique identifiers that you use to keep track of all items that need to be independently identified, stored, tested, reviewed, used, changed, delivered and/or maintained

- **Configuration control** – involves controlling what changes are made to the configuration baseline and when and how they are made.

- **Configuration status accounting** – involves keeping track of the state of all configuration items, all pending proposed changes, and all approved changes to configuration items.

- **Configuration audits** – involves ensuring that:

    - The configuration item performs as its design and specification indicate that it should perform

    - All configuration items for the system (or any of its subsystems) actually exist before the system (or subsystem) is accepted for testing or for production use

Configuration Management and ITS Systems

Configuration management's purpose is to keep the physical implementation of a product consistent with the documentation that describes how to build it and what it is supposed to do. By keeping the product and all its associated documentation synchronized throughout the development cycle, manufacturers reach the production stage of a product life cycle ready to begin mass-producing it. Thus, it has value for mass-produced items. Why does it also have value for "products," such as ITS systems, that aren't mass-produced?

Some authors describe configuration management's value in the design and implementation of ITS systems when "we start to invest significant resources in hardware and software development so further changes in requirements … start to get very costly in terms of budget and implementation time." This recognizes the value of good requirements to quality ITS system development. Managing changes to requirements is essential to minimizing cost and schedule overruns on ITS projects. Even if there were no other use for configuration management on an ITS project, its ability to control changes to requirements would be reason enough to employ it. Smith and Smith[3] provide testimonials on configuration management from several respondents and describe configuration management's use in a DOT's system, as a systems engineering tool.

[3] Smith, Brian L. and Bayne E. Smith, "An Evaluation of the Need for Configuration Management in Transportation Management Systems," The Transportation Research Board, November 2000

Configuration Management Costs

Configuration management doesn't come free. One cost associated with it is the cost of the configuration management system itself. The configuration management systems market place changes frequently. The International Council on Systems Engineering (INCOSE) tries to keep current on all systems engineering tools available to practitioners and has a website (http://www.incose.org/tools/tooltax/cm_tools.html) that attempts to keep a current and accurate database of available products.

Another important cost is that of administering the configuration management system itself. The contractor on an ITS project may handle the project's configuration management responsibility, but the public sector project manager must plan for this and incorporate these costs into his or her initial budget submission. When the project is over, however, the configuration management responsibility for the ongoing system becomes the government's. Configuration management is an ongoing need as the system evolves and requires maintenance over time. The public sector agency must integrate its ongoing configuration management needs into its organizational structure and make sure that its contractual agreement allows it to get the configuration management data that the contractor maintained.

What if You Don't Use Configuration Management?

You can avoid the costs associated with configuration management by not bothering to employ it on your ITS projects. If you do, you'll probably pay instead in costs for:

- Figuring out which system components to change when requirements change
- Re-doing an implementation because you implemented to meet requirements that had changed and you didn't communicate that to all parties
- Losing productivity when you replace a component with a flawed new version and can't quickly revert to a working state
- Replacing the wrong component because you couldn't accurately determine which component needed replacing

In addition, there's another potential cost. If you have a system that affects safety, configuration management can help you avoid mistakes that would endanger someone. For example, consider the consequences of fielding a new or upgraded collision detection radar with a subtle "bug" that fails to alert drivers about potential collisions when certain conditions of rain, fog, or snow occur. It might not be a problem; if the specific climatic conditions don't occur where the vehicle is being operated. In areas where (and when) they do, it could lead to loss of life.

The reason that configuration management is included as a key systems engineering practice is simple. It works! It keeps you from incurring costs you need to incur. And,

good systems engineers have learned, through practical experience, that it pays for itself many times over.

The lesson to learn is simple: Don’t pay the price later! Use configuration management.

Introduction

Purpose of This Document

This document is one in a series of monographs on systems engineering topics developed to help introduce this discipline to ITS project managers and their staff. The other documents in this series are:

- *Developing Quality Intelligent Transportation Systems Through Systems Engineering*, which is an overview of the entire systems engineering area

- *Developing Functional Requirements for ITS Projects*, which focuses on how to establish good requirements to drive an ITS project to its successful conclusion

- *Understanding Software Development: A Primer for ITS Public Sector Managers*, which introduces software development and its issues to public sector ITS managers who may have had little or no experience with software development

While none of these documents will turn the reader into an experienced practitioner in the area of systems engineering, they are intended to give their audience a sense of how the topic covered applies to Intelligent Transportation Systems (ITS).

This monograph covers the topic of *Configuration Management*. Although other monographs in this series discuss configuration management briefly, none goes into the topic at the same level as this one.

Intended Audience

Our intended audience includes:

- ITS project managers
- ITS project staff
- Contractors and their staff working on ITS projects
- Anyone else interested in applying configuration management, 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. Configuration management is a key discipline applied in good systems engineering practice. It's a primary tool used to ensure that systems get built for their maintainability and long-term health.

Origins of Configuration Management

Although some writers on configuration management claim it had its start with Eli Whitney's invention of the cotton gin (which was built with standardized parts), most believe that configuration management, as a systems engineering discipline, began in the 1950s with the building of complex military systems by the Department of Defense (DoD). The first documents describing configuration management and how it should be applied during the construction of products were military standards issued by the DoD.

In the 1950s, when the U.S. began developing tactical and strategic ballistic missiles, U.S. manufacturers of these weapons found that they had difficulty in mass-producing missiles for which they'd built successful prototypes. The major reason was that the manufacturers had failed to record the contents of the successful configuration in an organized manner. The first published military configuration management standard, ANA (Army, Navy, and Air Force) Bulletin 390, addressed the engineering change proposal for military aircraft, but could be applied to missile systems as well. It formalized the process of making changes to a system under development so that the builders maintained a record of the parts making up that system, what they did, and how they were tied together, from beginning to end. Thus, once the final prototype was accepted for production use, the weapon could be mass-produced. Over time, the processes and tools used for configuration management have changed, but the basic principles have not.

Configuration management has become an accepted practice in the manufacturing and software development industries. We'll discuss its value to these industries later in this document.

Current Configuration Management Standards

There are two key standards that cover configuration management in a broad context. The first is from the International Organization for Standardization (ISO) ISO 10007:1995, *Quality management – Guidelines for configuration management*. As indicated by its title, it is part of the quality control series of international standards. The second is the American National Standard Institute (ANSI) and Electronic Industries Alliance (EIA) joint standard ANSI/EIA 649-1998, *National Consensus Standard for Configuration Management*. Both of these documents are equally applicable to hardware or software configuration management.

The Institute of Electrical and Electronics Engineers (IEEE) also has a standard relating to configuration management, IEEE Std 828-1998, *IEEE Standard for Software Configuration Management Plans*. This standard is part of the IEEE's software engineering standards and focuses on software configuration management.

The two general standards for configuration management provide you with guidance on how to employ configuration management on your project. They describe the kinds of things that your configuration management system should do and relate configuration

management to quality standards, such as the ISO 9000 series of documents. However, they don't specify a process. For that, you may want to look at some of the books listed in the References section of this document, or similar material. If you are the public sector project manager for an ITS project, you probably won't set up the configuration management system; you'll have your integration contractor (or some other contractor on the project) set it up and run it. But these standards help you understand what should be going on, so that you can make better informed decisions about the process and how it's working.

The IEEE's software engineering standard provides more specific guidance on what you should plan to do for software configuration management. If you are the public sector project manager for an ITS project, you should look at this document to see what your software development contractor should give you in a software configuration management plan for the project.

Now, let's discuss what configuration management involves.

Configuration Management Principles

The ANSI/EIA standard for configuration management[4] defines configuration management as: "A management process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design and operational information throughout its life." This involves having a document (or series of documents) that describes the components that make up any version of a product (hardware or software), along with the requirements that the version of the product is intended to satisfy. The documentation includes diagrams or descriptions of the product's design, its operating parameters, and anything else required to ensure that interchangeable copies of the product can be produced at any time. On an ITS project, the "product" is the system that you're implementing.

Configuration management involves five major areas:

- Configuration management planning
- Configuration identification
- Configuration control
- Configuration status accounting
- Configuration audits

The following sections discuss each of these areas in more detail.

Configuration Management Planning

Configuration management planning involves making decisions about what needs to be controlled within a product configuration, when you establish a controlled configuration, how you change a controlled configuration, and what amount of effort you're going to expend to manage configurations. The decisions you make are formalized in a *configuration management plan*, which becomes part of the documentation for the product or system that you are building. This plan explains how you will control the configuration of the system you are building.

Deciding What Configuration Items to Control

If your system, like most ITS systems, consists of both hardware and software items, you must decide the level at which you will control these items. For hardware, you can choose from among the following levels[5]:

- Part – one piece (or two or more joined together pieces) not normally subject to disassembly without destroying or impairing the part's designated use. Example: a processor chip

[4] ANSI/EIA 649-1998, *National Consensus Standard for Configuration Management,* p. 4
[5] Adapted from Buckley, *Configuration Management: Hardware, Software, and Firmware*, pp. 7-9

- Subassembly – two or more parts that form a portion of an assembly or a unit replaceable as a whole, but having a part or parts that are individually replaceable. Example: a printed circuit board

- Assembly – a number of parts or subassemblies (or any combination thereof), joined together to perform a specific function. Example: a traffic signal controller

- Unit – an assembly or any combination of parts, subassemblies, and assemblies mounted together, normally capable of independent operation in a variety of situations. Example: a traffic signal controller cabinet

- Group – a collection of units, assemblies, or subassemblies that is a subdivision of a set or system, but is not capable of performing a complete operational function. Example: time-based coordinated system

- Set – a unit or units and necessary assemblies, subassemblies, and parts connected or associated together to perform an operational function. Example: closed-loop system

- Subsystem – a combination of sets, groups, and so on that performs an operational function within, and is a major division of, a system. Example: Regional automated traffic control subsystem

- System – a combination of parts, assemblies, and so on, joined together to perform an operational function or functions. Example: an automated traffic system

Figure 1 illustrates these hardware levels.

The hardware level at which you manage your configuration depends on your maintenance plans or expectations for the future. If you will maintain the system by diagnosing and replacing failed chips, or if you anticipate chip upgrades in the future, you'd manage your configuration at the *parts* level. If you plan to perform "board level" maintenance, you'd manage your configuration at the *subassembly* level. This is the same level you'd use if you anticipate future "board level" upgrades or if, as in the case of a type 2070 traffic signal controller, different controller assemblies have different boards, depending on what function they perform.

If you're going to install and maintain hardware as a complete assembly and treat controllers as "black boxes," you can manage your configuration at the *assembly* level. This is the level you'd use if knowing the model number of a device is sufficient, since you're going to maintain the devices by model number.

You could also choose to manage your configuration at the *unit* level. For a traffic signal system, this would mean tracking the entire set of intersection equipment (controller, flasher, conflict monitor, load switches, etc.) as a single element in your configuration

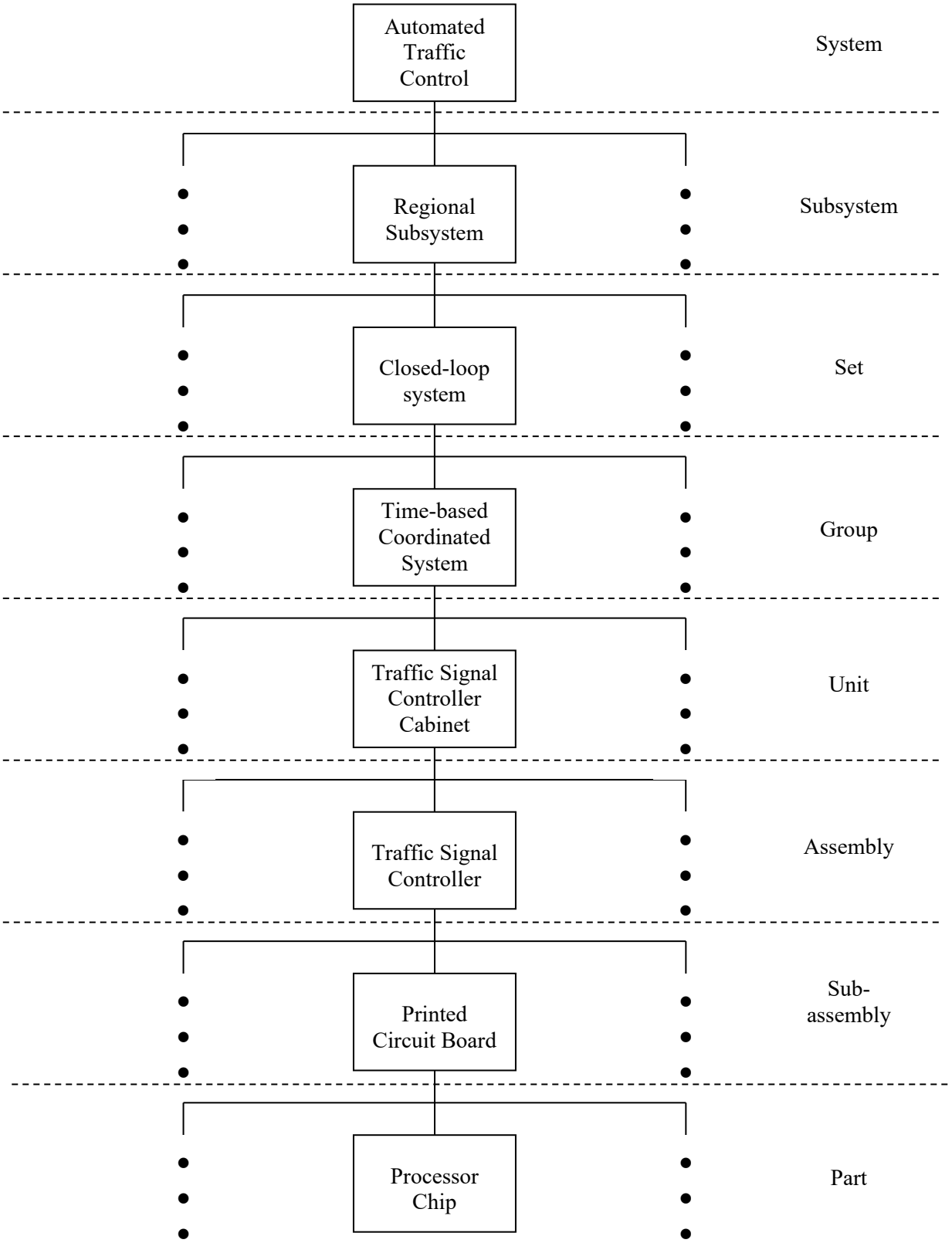


Figure 1
Hardware Levels for
Configuration Management

management system. If this level is adequate for your operation and maintenance of your system, then you can manage your configuration at this level.

The other levels of hardware are inappropriate for managing the configuration of most ITS systems.

For software, you establish control over the software configuration items at different levels, depending on the type of software that you're managing. For *application programs*, i.e., the software that performs the "business" functions of your system (where "business" refers to the mission of the system), you need to control the software configuration items at the individual program component level. And your will need to control two kinds of application programs: *source code*, which is the program in the original programming language, and *executable code*, which is the program in the "machine" language of your computer. For *commercial off-the-shelf (COTS) software*, i.e., software you acquire from a vendor and use (more or less) in the form that you receive it, you may only need to control the software at the product (system) level. This type of software includes:

- Operating systems (OS)
- Database management systems (DBMS)
- Geographic information systems (GIS)
- Development tools, e.g., compilers, computer-aided software engineering (CASE) tools
- Test tools

Upgrading Desktop Software

Pick up a shrink-wrapped box of your favorite software (or some new piece of software you'd like to try) at your local software store and look at the box it comes in. Usually, on one of the box panels, you'll find information about the hardware and operating systems required to run the software. For example, it may say that it runs on "Windows ME/2000/NT." It may require a "Pentium II 266 Mhz processor, 64 Mbytes RAM, and 20 Mbytes storage available." It may need a "4X CD-ROM." What this tells you is that the manufacturer of the software certifies it to work on certain operating systems and has indicated the minimum hardware requirements to run the software. If you have a faster processor, a faster CD-ROM drive, more memory, and more free disk space, you can run the software. But if any of your hardware is less capable, smaller, or slower than that required, the software probably won't run. The issue is less clear when we're dealing with the operating system. If you have a later operating system than the one listed by the software manufacturer, the software may run on it, but there's no guarantee.

Still using the PC example, if you had software that ran under DOS, it wouldn't run directly on a Windows operating system. It would still work if the Windows operating system lets you run your computer in 'DOS mode.' But the latest version of Windows, Windows 2000, no longer has a 'DOS mode.' Therefore you can't run DOS software if you've upgraded to that version. In fact, sometimes DOS software wouldn't run on later versions of DOS, because the programs checked for the DOS version number and only executed if they recognized the version number.

Program incompatibilities with hardware and software occur because each manufacturer is upgrading its product to meet changing demands and is not necessarily concerned with maintaining compatibility with older operating systems or hardware platforms.

In the business world, companies have to consider hardware and software compatibilities when they upgrade applications, operating systems, and database management systems. If your automated traffic control system is only certified to run under Oracle's relational database management system (RDBMS) version 7.4, you may not want to upgrade to Oracle RDBMS versions 8i or 9i and take the chance that the software will no longer function properly.

For software configuration items that you are controlling at the component or program level, you need to know the item's unique identifier (name or number), the physical directory or storage location where the item is kept, and any other information (e.g., version number) that uniquely identifies the program. You also need to know what software libraries are used with the program, specifically identifying the components from each software library that get compiled with the program. For software configuration items that you are controlling at the system level, you must know the physical directory or storage location where the system is kept, the version number, and what, if any, associated "patches," program modifications that the system's vendor supplies to correct problems with the software, have been applied.

Some COTS software items also have relationships with the hardware on which they run and with each other. For example, for specific models of computer hardware and operating system versions, you may need to have a specific version of a DMBS or GIS installed. Later versions of the DBMS or GIS software may not run as effectively on a particular combination of hardware and operating system, if the COTS software was optimized for later hardware/OS combinations.

Another area to consider is documentation. Documents contain the written record of your system as it evolves. The types of documents you should consider controlling include:

- System requirements
- Interface control documents
- High-level design documents
- Detailed design documents
- Hardware technical data packages
- User manuals
- Maintenance manuals
- Program specifications
- Test plans
- Test procedures
- Test reports
- Test data documents[6]

Every controlled document must have a unique identifier, so that it can be inventoried and controlled. In addition, each copy of a controlled document should have its own copy identifier, e.g., "copy ***x*** of ***n*** copies," so that you can keep track of who has copies of controlled documents. When a document changes, you must send copies of the changes to all holders of copies of controlled documents. (It helps to have dates associated with each page of a controlled document. Then, one can always verify that one has the latest version of a controlled document by comparing the dates in one's copy to the dates in the Master copy of the document.)

[6] The test data themselves, in electronic form, should also be controlled, to ensure that tests can be repeated as necessary.

Other elements that you should consider when deciding what to put under configuration management include:

- **Communications interconnections** – think about maintenance issues related to your wide area networks (WANs) and local area networks (LANs). For example, if you are using static IP (Internet Protocol) addresses on your local area network, you can't issue the same IP address to two or more devices on that network. Doing so causes address-resolution conflicts and leads to network problems. Managing IP addresses through your configuration management system helps you avoid these conflicts. Other questions you might ask include: What are you going to need to know to operate and maintain these networks effectively? Do you know how data ports are connected to routers or bridges so that you can trace and diagnose network issues when they occur? Questions like these help you decide what configuration items to manage.

- **Wiring** – do you know what wires or cables connect different pieces of equipment? Can you identify the wires or cables through some physical identifier that allows you to find a specific wire or cable in a (possibly tangled) collection of wires and cables? Do you know what frequency assignments have been made where multiplexing is used? Questions like these are important in deciding what configuration items to manage.

- **Numbering** – how do you link numbering schemes for detectors, communications links, and other elements in your network so that you can trace data from a detector in the field to a record in a database or to one of your displays? Can you trace the connections from one end of a communications line to another?

It's important to think about how you plan to use the information in your configuration management system after the system you're building is in operation. Consider the operation and maintenance aspects of that system and it'll help you decide what items you need to manage in a configuration management system.

Deciding When to Control Configurations

In configuration management, control over a configuration is normally established after the project reaches a *baseline*. Originally, *baseline* was an engineering surveying term that described a boundary line, with known direction and fixed end points, from which engineers extended maps into previously unmapped terrain. In systems engineering, we normally define six major baselines for a system, shown in Figure 2:

- Requirements baseline: the point at which the major requirements for the system have been defined

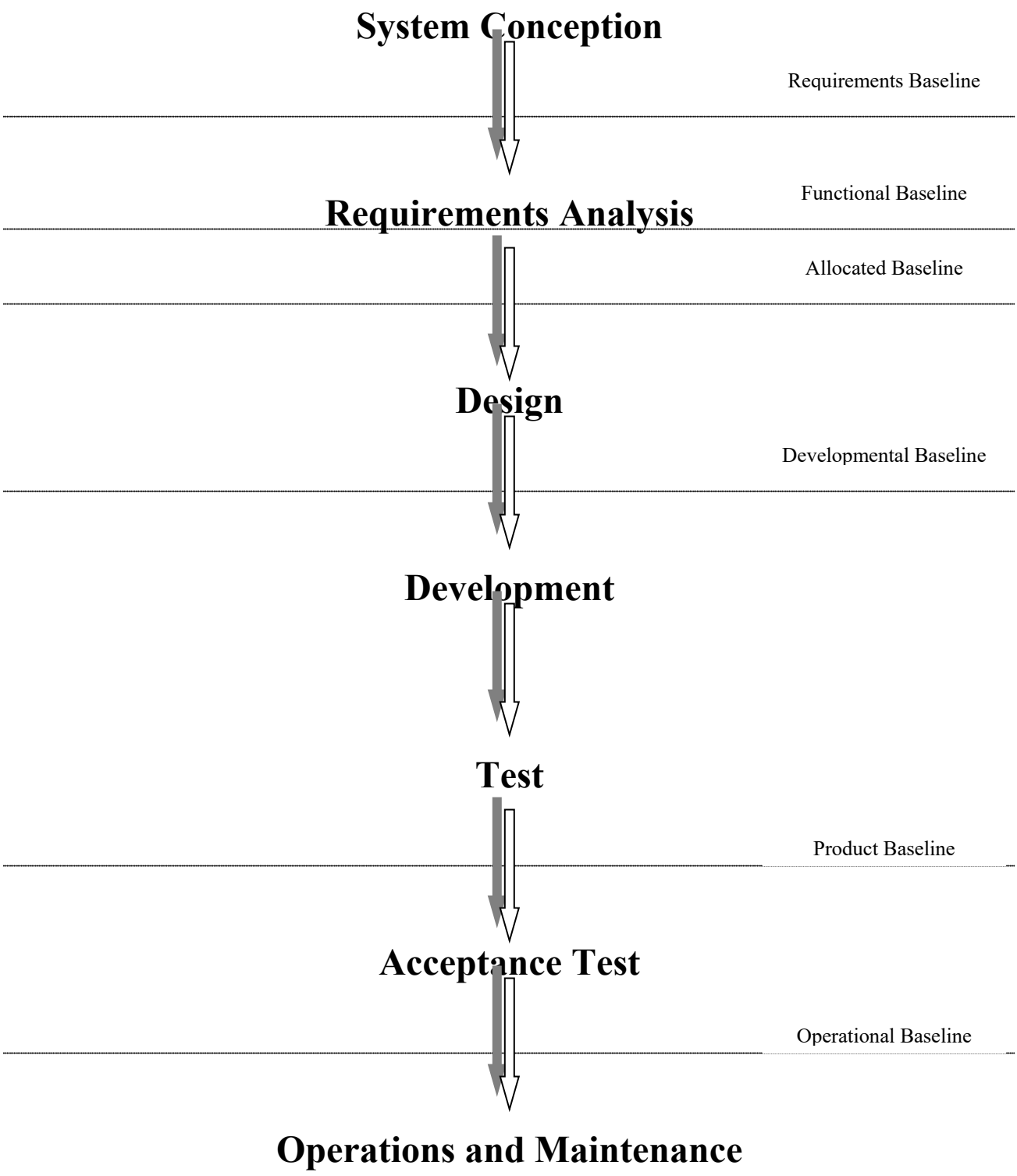


Figure 2
Baselines in the System Life Cycle

- Functional baseline: the point at which all of the functions of the system have been identified

- Allocated baseline: the point at which all of the system's functions have been allocated to the system's components in which they will be performed

- Developmental baseline: the point at which the system's design is complete enough to begin development

- Product baseline: the point at which the system is ready for installation

- Operational baseline: the final version of the system, as put into production use

It's important to note that a system, if it is developed in stages or phases, can have multiple baselines in place simultaneously. For example, one version of the system can be in its operational baseline, while a second version of the system – the next version to be fielded – could be at its developmental baseline. A third version of the system, one that's just being planned, could be only at its requirements baseline.

It's also possible to combine some of the baselines. For example, one could combine the functional and allocated baselines into a single baseline that described what the system was intended to do at a detailed level. One could also combine the product and operational baselines, if the system will not change between the time it is accepted for operational use and when it is actually put into operational use.

The documentation for any baseline consists of descriptions of the configuration items under configuration control, the configuration items themselves, and *all changes approved to the configuration items being controlled.*

Deciding How to Change a Configuration

You're not likely to build a system without having some changes to an established baseline. The goal of configuration management is not to keep changes from occurring, but to permit changes in a controlled fashion, ensuring that what is being built is internally consistent with its description at all times. At the same time, it's best to avoid unnecessary changes to configuration baselines, as any change can increase either the amount of time necessary to build the system or the cost of building the system, or both.

The configuration management plan should indicate who may propose changes to a baseline, and what kind of justification for changes is necessary. It should also state the process for proposing and justifying a change, describing any forms you want used for change proposals. In many organizations, there is a formal document, often called the Engineering Change Proposal (ECP), that is used to describe proposed changes to a system's configuration baseline. You may decide to use a different approach, but it's better to have some formality in this process. The formality in the process creates an audit trail of all changes considered as well as a record of those approved, those rejected,

and those deferred. This audit trail can come in handy in future assessments of proposed enhancements or changes to the systems.

Deciding on Configuration Management Resources

Although the ideal situation would be to embed configuration management practices as part of the normal project management process, frequently good configuration management requires project staff resources devoted to this effort, at least part time. On large projects, with many configuration items and many change proposals, configuration management can easily become a full-time job for more than one person on the project staff.

You need to decide upfront how much effort you're willing to spend on this activity and who will do it. You can use your own organizational resources or you can contract out the configuration management activity. Either way, there is a cost involved and you need to understand what it will be and plan for it.

Configuration Identification

The things within a configuration baseline that you decide to control through configuration management are known as *configuration items*. Kelly provides a good definition of what a configuration item is:

> "A configuration item (CI) is any part of the development and/or deliverable system (whether software, hardware, firmware, drawings, inventories and/or documentation) which needs to be independently identified, stored, tested, reviewed, used, changed, delivered and/or maintained. CIs differ widely in complexity and may contain other CIs in a hierarchy."[7]

Once you've decided what configuration items to control, you've just begun the effort. Each item that you want to control must have a unique identifier associated with it. The item's identifier for the item should be marked on it in some fashion (if possible), so that the item can be identified without error and tracked.

There are several reasons for this. First, you want to be able to know what items you have under configuration control and be able to locate them. Marking them with a unique identifier makes it possible to do that. Second, you want to be able to track the status of each item as it progresses toward completion. If there are changes to be made to an item, you want to know what its current status is and whether the change has been incorporated in it. Third, if a change made to a specific item of a group of like items makes it different from the other items it resembles, or if a unit (or higher level of hardware aggregation) contains a part, subassembly, or assembly that makes it different from other similar units, you want to be able to track these differences and any impact they may have on the system. For example, suppose you're upgrading the signal controllers in a traffic signal system, replacing older controllers with newer, more capable ones. If, during the

[7] Kelly, *Configuration Management: The Changing Image*, p. 36

upgrade, the manufacturer of the new signal controllers tells you that a chip set in some of the controllers is faulty, you'd want to know which controllers are affected and where they are, so that you could replace the faulty chip sets.

The latter reason, known as the "where used" problem, may require expanding the information kept on a particular configuration item to include data on what parts, subassemblies, and assemblies it contains. Many organizations require this when the failure of an incorporated part, subassembly, or assembly could have a significant effect on safety or cause serious social or financial losses. Airlines, for example, keep a record of key parts, such as engines, that make up their aircraft. Then, in case of a crash that is attributed to engine failure (or the failure of some other critical part), they can check out other aircraft that might suffer the same fate because a part with a similar history also fails.

The "where used" problem can affect software as well. Take the following two examples.

First, remember the "Y2K" issue that, back in the late '90s, was considered a major issue we faced? The concern was that some existing software wouldn't correctly handle dates when the first two digits of the century changed from '19' to '20'. One of the major aspects of that problem was determining where date-handling routines were used that assumed two-digit year values should be preceded by '19'. Identifying those software items was the first step in fixing potential problems. Then, after you'd identified the software items, you had to know which ones had been fixed. The third step was to replace the flawed item(s) with the fixed one(s). The effort involved in the next step depended on how many different copies of the same flawed date-handling routine were in use and where they were being used. In many cases, particularly for commercial products where an organization might use copies of the same product on multiple computers, once all of the Y2K bugs in the software were fixed, organizations still had to ensure that they replaced each copy of the flawed software with a corrected version. This meant being able to link the flawed software product to all instances where it was installed.

Second, consider the case of a traffic signal control system where each controller is running software that manages the signals it controls. Whenever you replace that software, you have to go in and install a new (updated) version of the software in each controller. If you find, during the process of replacing the software, that the new software has a critical "bug," you have one of two options: 1) quickly get a corrected version of the software and replace all the "buggy" versions as well as the older versions[8]; or 2) re-install the older versions of the software in all controllers that have the new, buggy software and wait until you get a corrected version of the software from the vendor – then you can install the correct version in all your controllers. Regardless of which option you choose, you need to know where the flawed software is being used. If you don't know where you've installed the buggy software and thus don't replace it with software that works properly, you are incurring a latent liability.

[8] If the bug was a safety-related one, you wouldn't normally choose the first option.

Exactly what detail you need for appropriate configuration identification depends on the specific needs of your system and your operational needs. At a minimum, you need to have all hardware and software items (and you have to define what these "items" are) identified and controlled, if you're going to do effective configuration management. But consider the "where used" issue and determine if it applies to your system. Don't get caught short.

Configuration Control

The principal concern in configuration control is controlling what changes are made to the configuration baseline and when and how they are made. In the ideal world, you could "freeze" a configuration baseline and permit no changes until the product or system was completed. In the real world, "freezing" a baseline is impractical. To freeze a baseline, you'd have to be sure that no changes to it would be needed to make it work properly. In reality, since fallible human beings build systems, change is common, mostly to correct mistakes that people have made in the design or development of system components.

But even though changes are generally needed, you shouldn't make any change to an approved baseline until you've assessed and evaluated its potential impact. The process of proposing, assessing, evaluating, reviewing, and deciding whether to approve or reject a proposed change is *configuration control*. This is probably the most critical area of configuration management, because it can most strongly affect the success of the project.

Proposing Changes to a Configuration Baseline

It's commonly said that, "Anyone can propose a change through configuration management." Well, you decide who "anyone" can be when you prepare your configuration management plan. In the developmental stages of a system, the designers and developers are usually the ones who are proposing changes. If your system developer is a contractor that you've hired, the contractor proposes changes to the system design or to system components under development when the contractor believes that the change enhances the chances of meeting the project schedule or controlling the costs of building the system. Contractor personnel also propose changes when they believe that there is some new technology or approach whose incorporation would enhance the system. Frequently, when the contractor proposes a change, the contractor also provides an assessment of the overall impact of that change on the project, i.e., what its effect will be on the project schedule and/or project cost. It is also possible for users to propose changes during system development stages, usually because they perceive a change (or error) in their requirements for the system.

If a change corrects an error in a system requirement or system component, there is a strong case for approving it. Regardless of the cost of fixing the change at the time it is proposed, it is usually significantly cheaper to fix a mistake during the developmental stages than after the system has been implemented. Studies have shown that the cost of

fixing a mistake goes up as much as 10 times with each subsequent stage in the system life cycle. However, if the change is an enhancement to the system, an expansion of the scope of work of building the system, there is a strong case for disapproving it. Generally speaking, enhancements can be and should be postponed until you've operated the initial version of the system for enough time to judge its capabilities and decide what improvements are really needed. If someone proposes a change just to insert new technology into the system, this proposal should be studied carefully. While new technology can provide cost savings, it also brings risk with it, particularly if it is unproven technology. It may be better to wait until the system is implemented and you are ready to enhance it before introducing something new and unproven.

Reviewing and Approving Changes to Configuration Baselines

Most configuration management systems require a specific format for submitting a change proposal. In many cases, this format is known as the Engineering Change Proposal (ECP). Figure 3 illustrates a sample ECP. The data that should appear on an ECP include:

- Name of the individual proposing the change
- Contact information (e.g., telephone number) for the individual proposing the change
- Configuration item(s) for which the change is proposed (may either be the identifying number(s) or a description of the item(s) if the proposer doesn't know the identifying number[s])
- Description of the proposed change
- Priority/Importance of the proposed change – this can be an error severity level, e.g., "fatal," "serious," "moderate," etc., or a priority scheme, such as "immediate," "high," "medium," "low"
- Description of the impact if the change is **not** made (proposer's judgment)

You can include other data in the format of the ECP (or whatever name you use for your change request), but these are the minimum data to ask for.

The next step in the process is for someone to review the change and decide what impact making it will have on the system and the project. There are two kinds of reviews that may need to be done: a technical review and a business review. The *technical review* considers the technical aspects of the proposed change:

- What configuration items (other than the one identified in the ECP) does it affect?
- Is the change technically feasible?

- What will this change involve in terms of modifications to the different configuration items affected?

- How much effort (labor) will be involved in making the change?

- How much will making the change cost in additional project costs?

- What delays, if any, will this cause in the project schedule?

- How severe an impact, from a technical point of view, will **not** making the change have on the system and the project?

- Are there "work arounds" that could be used to postpone or avoid making the change? If so, are these practical?

The *business review* considers similar types of questions:

- What business processes are affected by this change?

- Does the proposed change make sense from a business point of view?[9]

- Will this change improve business processes? Or does the change fix a problem within an existing business process?

- Will current staff resist this change?

- How severe an impact, from a business point of view, will **not** making this change have on the system and on the project? (This is an assessment or validation of the justification provided in the ECP.)

The developer on the project normally carries out the *technical reviews*, as that organization usually has the best technical overview of the project. However, user personnel, from the system's customer organization should carry out the *business reviews*. These are the people who best understand the business impact of a proposed change.

[9] In this context, "business" refers to the mission of the government organization involved. If, for example, the mission of the government agency were as simple as "provide accurate traveler information on traffic conditions," the questions asked about the change would relate to how it affected the agency's ability to provide accurate and timely information on traffic conditions along its area of responsibility.

Engineering Change Proposal

ECP No:	ECP Title	Date Submitted:
Originator Name and Address:		**Configuration Items Affected:**
Originator Telephone No:		**Priority:**
Description of Proposed Change:		
Reason for Change/Impact of Non-Incorporation:		
Potential Risks of Change:		
Expected Cost of Change:		**Expected Duration:**
Disposition ☐ Approved ☐ Disapproved ☐ Deferred	**Disposition Date:**	**Disposition Authority:**
Comments		

Figure 3
Sample Engineering Change Proposal

If the project is large and complex, the people who do the reviews may be organized into standard teams and assigned permanent configuration management responsibilities on the project. Otherwise, they may be convened to do these reviews on an *ad hoc* basis. You'll need to decide which approach to take during your configuration management planning. If the teams are organized as permanent review teams, then the project's configuration management organization might appear as depicted in Figure 4.

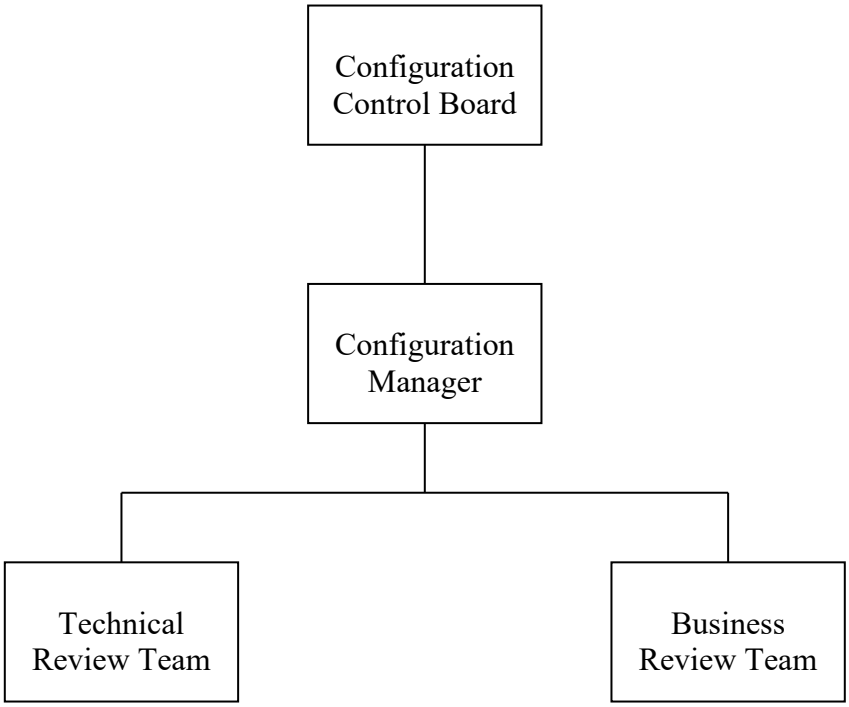


Figure 4
Sample Configuration Management Organization

We've already discussed the roles of the two review teams. Let's consider the other two boxes on this organization chart.

The *Configuration Manager* is the individual to whom you assign overall responsibility for the *mechanics* of configuration management. He or she is responsible for defining, implementing, and maintaining the configuration management system for the project and the system (or product) being built. This does not have to be a full-time role for the individual, although on projects with large numbers of configuration items that need to be controlled it could be a full-time job[10]. This individual makes sure that all configuration items are identified, that their status is kept up to date and all status reports are prepared and distributed to the appropriate parties, that configuration audits are done as needed,

[10] Some projects are large enough to have separate individuals, reporting to the Configuration Manager, assigned responsibility for 1) Documentation, 2) Hardware, and 3) Software. In those cases, it may also be necessary to have an individual who is the *Configuration Control Clerk*, who handles all of the administrative paperwork involved in very complex configuration management on very large projects. In the ITS world, such projects will be rare.

and – very important – that all proposed changes are reviewed and evaluated and a report on them, with a recommendation on whether they should be approved, is prepared for the *Configuration Control Board*.

The *Configuration Control Board* is a critical part of configuration management. This is the group that is empowered to review and *approve* or *reject* all changes that have a material effect on the overall project schedule and costs. ("Material effect" means that either the project schedule gets longer or the project cost increases or both.) On ITS projects, the chair of the Configuration Control Board is usually the Government Project Manager, since that individual has overall responsibility for project schedule and budget. In addition, the Configuration Control Board should include at least one key user representative (to bring the "business" focus to the board), and a Contractor representative, usually the Contractor's project manager. Others who might serve on the board include a senior public sector with funding responsibility (because of the potential for cost impacts on the project), and the project's Configuration Manager.

Figure 5 illustrates how the configuration management process might work. The Configuration Manager must ensure that all change proposals are reviewed and evaluated, with the evaluation including a recommendation to the Configuration Control Board on whether they should approve or reject the change. It is the Configuration Control Board's responsibility to make the decision to accept or reject the change proposal. Once they make the decision, it then becomes the Configuration Manager's responsibility to communicate that decision and track its implementation. If the change proposal is rejected, the Configuration Manager informs the originator of the result; if the change proposal is accepted, the Configuration Manager informs the originator and also passes the approved change proposal over to the appropriate project team for implementation. The Configuration Manager tracks the status of all approved change proposals until they are incorporated in the relevant configuration item(s).

Configuration Status Accounting

Configuration status accounting means keeping track of the state of all configuration items, all pending proposed changes, and all approved changes to configuration items.

The status accounting process is mostly one of reporting on what progress the project team has made in completing configuration items and in incorporating approved changes into configuration items. Originators of ECPs are interested in the status of their proposals, so they will want to see reports giving the status of pending ECPs. But, although the configuration status accounting process may seem trivial, there is one important function that it plays. It provides a very useful metric on the quality of specific configuration items.

A truism in system development is that, if a specific item under development has several problems that have led to ECPs to fix errors, then there are probably additional errors in that item that have not yet been found. The configuration status accounting process can lead to early detection of potential problem spots in the system design, if there are

clusters of ECPs, related to “bug” fixes, around specific configuration items. The Configuration Manager should be alert to this type of issue and raise it to the project manager should it occur.

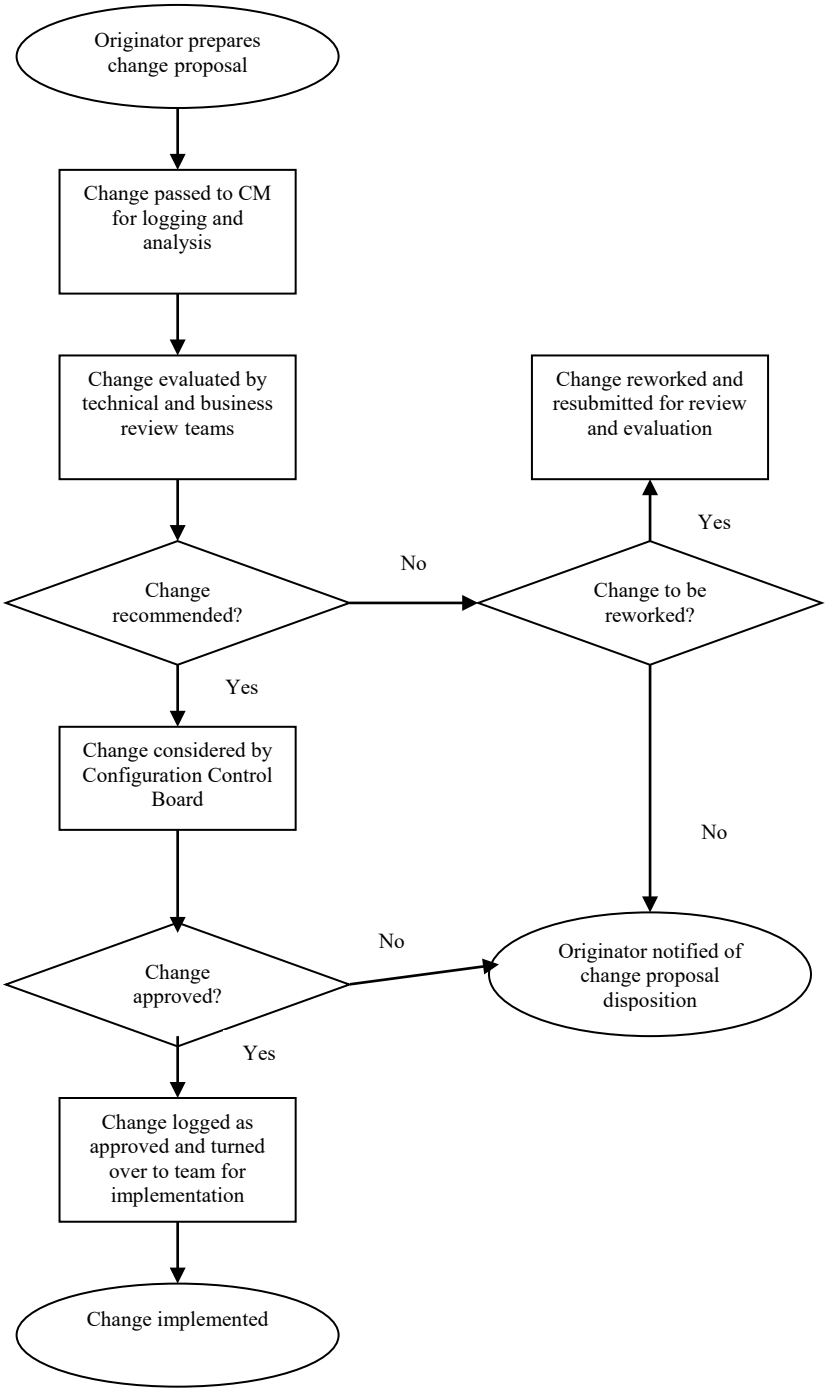


Figure 5
Change Control Process

Configuration Audits

There are two basic types of *configuration audits*: the Functional Configuration Audit (FCA), and the Physical Configuration Audit (PCA). The Configuration Manager or his representative conducts a FCA to make sure that the configuration item performs as its design and specification indicate that it should perform. FCAs should be conducted prior to the beginning of any test phase on the project. A PCA is an inventory of the configuration items for the system (or any of its subsystems) and ensure that all of the items are actually in existence before the system (or subsystem) is accepted for testing or for production use.

The Configuration Manager, in conducting a PCA, should rely on the status reports for the baseline being audited to know what items it should include.

We'll that's a quick tour of what configuration management's all about. Now let's look at how configuration management fits into the ITS context.

Configuration Management and ITS Systems

The basic purpose of configuration management is to keep the physical implementation of a product consistent with the documentation that describes how to build it and what it is supposed to do. During product development, manufacturers use configuration management to track the evolution of a product and its design. This is true whether the product is a physical thing or software. By keeping the product and all its associated documentation synchronized throughout the development cycle, manufacturers reach the production stage of a product life cycle ready to begin mass-producing it and packaging it for sale along with its instruction and maintenance manuals.

Thus, configuration management has value in cases where the product will be mass-produced. Why does it also have value for "products," such as ITS systems, that aren't mass-produced?

McQueen and McQueen point out configuration management's value in the design and implementation of ITS systems as follows:

> "At this point [design and implementation] we start to invest significant resources in hardware and software development so further changes in requirements will start to get very costly in terms of budget and implementation time. There is also no point in kidding ourselves that the requirements are completely set in concrete. There may be legitimate reasons discovered once the implementation is under way for changing requirements. Having recognized that this is the case, it is absolutely vital that some mechanism be established to manage this aspect of the implementation.
>
> In our experience, many of the problems associated with cost and time overruns in ITS implementations have their origin in this issue. While recognizing that a few late changes are inevitable, these need to be managed and subjected to a high degree of discipline. The system engineering profession has identified this as an important issue and developed management techniques to address it. These are referred to as *change control* or *configuration management procedures*." [11]

This quote recognizes that good requirements are critical to quality ITS system development and that the issue of managing changes to requirements is essential to minimizing cost and schedule overruns on ITS projects. Even if configuration management had no other use on an ITS project, its ability to control changes to requirements would be reason enough to employ it.

Smith and Smith also attest to the value of configuration management in ITS. Their work is based in part on a survey they conducted in the spring of 2000 to determine whether and how State Departments of Transportation (DOTs) used configuration management in transportation projects. They received responses from 36 State DOTs and found that "as systems get larger and more complex, the use of configuration management becomes

[11] McQueen and McQueen, *Intelligent Transportation Systems Architectures*, pp. 293-4

more commonplace." [12] In addition, they report the following three testimonials on configuration management as representing strong support from many respondents:

> "With almost 20 years in the design, implementation, modification and expansion of our system, the benefits of being quickly able to recover by returning to an earlier working state are enormous. Our system has been very dynamic, and there is always some area where we are working on improvement or upgrade, while still actively managing traffic."
>
> "As in any large, complex system, configuration management can provide a constant understanding of the current state of the system…. The key factor in configuration management is having a central repository of information for reference as personnel changes occur over the life of the system. It is also a great aid in maintaining the system when items are replaced for repair. Technicians should have ready access to configuration data when installing or re-installing standard system components."
>
> "A formal, documented configuration control process can save operational costs over the life of the contract and mitigate the impact of personnel and equipment changes."[13]

The Smiths also include a case study of configuration management's use in the Georgia DOT's (GDOT's) Navigator system, which GDOT developed to manage transportation in Georgia for the 1996 Olympic Games. The Utah DOT (UDOT), which is adapting Navigator for use during the 2002 Winter Games in Salt Lake City, also is using configuration management to handle its changes to the Navigator system. Both states are strong believers in its value as a systems engineering tool.

So, given that configuration management is important, how do we use it? First, let's consider how we might use configuration management during the development of an ITS system.

Using Configuration Management During ITS System Development

In ITS system development, as in all good systems development, the place to start is with requirements. Say we're building a system that involves traffic monitoring. We might write some requirements for the traffic-monitoring portion of the system as follows:

R9 The system shall have the ability to monitor traffic volume
- R9.1 The system shall measure traffic volume
  - R9.1.1 The system shall measure traffic volume in 5-minute increments
- R9.2 The system shall measure traffic speed

We're numbering our requirements uniquely, prefixing each with the letter "R" and getting more detailed about what's required the more digits to the right we add to the requirement number.

---

[12] Smith and Smith, "An Evaluation of the Need for Configuration Management in Transportation Management Systems," p. 15.
[13] *Ibid.*, pp 15-16.

At some point in the design process, we transform requirements (statements about *what* the system must do) into specifications (statements about *how* the system will perform the action that satisfies the requirement). So we might transform requirement R9.1.1 into the following specification:

> S38.17 The system shall calculate traffic volume by summing the actuations of each detector over a five-minute period.

Here we're numbering specifications with an "S" prefix. The unique numbers we assign to each requirement and each specification will aid us in tracking the relationship among requirements and other elements of the system and its design.

In turn, we might decide to implement this specification using three separate software modules, as follows:

- Detector Data Module 31 (DDM031) – records each detector actuation that occurs, time-stamps it with date and time, down to the nearest second, then transmits the data to a summarizing module

- Volume Summary Module 22 (VSM022) – summarizes all actuation data that it receives during each discrete five-minute interval. At the end of each interval, it passes the summarized counts to a data storage module, clears its counters, and starts over again.

- Data Storage Module 40 (DSM040) – takes the summarized traffic volume data and records it, along with the data and start time of the five-minute interval, in a traffic volume database.

Each of these modules has a unique identifier as well. In our example, we've used a three-letter prefix to identify the area in which the module functions and then added a three-number suffix to distinguish each module from others in the same area.

It's important for us to know how requirements relate to other system elements, so we create a traceability matrix that shows the interrelationships. One row in that matrix might look like Figure 6.

Requirement	Specification	Software Modules
R9.1.1	S38.17	DDM031, VSM022, DSM040

Figure 6
Traceability Matrix Row Example

At some point in our development effort, we'll create tests to determine whether the system meets the defined requirements and how well it meets them. We need to link

each test defined to all requirements that it verifies and validates. Let's say we define the following test to determine whether we've met Requirement R9.1.1:

> Volume Test 63 (VT063) – This test will run for 12 minutes, falling into three discrete five-minute increments:
>
> - Increment #1 – From T+2:00 to T+4:59: 17 vehicles will pass through the detection area
> - Increment #2 – From T+5:00 to T+9:59: 42 vehicles will pass through the detection area
> - Increment #3 – From T+10:00 to T+13:59: 28 vehicles will pass through the detection area

This is a controlled test, with known quantities of vehicles passing through the detection area, to determine how well the system performs and whether the system does calculate traffic volumes correctly for each five-minute period. We correlate this test to the other elements in our matrix by expanding the row as shown in Figure 7.

Requirement(s)	Specification(s)	Software Module(s)	Test(s)
R9.1.1	S38.17	DDM031, VSM022, DSM040	VT063

Figure 7
Expanded Traceability Matrix Row Example

OK. Now we've established traceability. How does this relate to configuration management?

Studies have shown that system requirements change, on average, at the rate of 1-2% per month of system development effort. That means that, on a system development project that takes 10 months, between 10% and 20% of the total system requirements will probably change. Let's assume that our detailed requirement changes so that we measure volume in 10-minute increments rather than 5-minute increments. How does this change ripple through our system?

Since we've established traceability from the requirement to other parts of the system, we can find the related pieces. The first thing we have to do is change the requirement. Thus, the detailed requirement gets re-stated as follows:

> R9 The system shall have the ability to monitor traffic volume
> R9.1 The system shall measure traffic volume
> R9.1.1a The system shall measure traffic volume in 10-minute increments
> R9.2 The system shall measure traffic speed

We re-number the requirement by adding an "a" suffix to the original number to indicate that it's been modified. We'll follow a similar procedure with the other elements that are related to this requirement, if we modify them.

We modify the associated specification by changing the duration of the interval during which the traffic volume count is made, as follows:

> S38.17a The system shall calculate traffic volume by summing the actuations of each detector over a ten-minute period.

So far, all we've changed are paper records. Now comes the time when we need to change a software module. We know that there are three software modules associated with this requirement, but we don't have to change all three. It may only be necessary to change one: VSM022. This module sums traffic data sent by the data detection module. In its "current" version, it does this for a five-minute interval. The modified program would have the following description:

> Volume Summary Module 22 v2 (VSM022 v2) – summarizes all actuation data that it receives during each discrete ten-minute interval. At the end of each interval, it passes the summarized counts to a data storage module, clears its counters, and starts over again.

In this case, rather than adding an "a" suffix to the identifier, we note that the software module has a different version.

As long as the software module that writes the database record isn't responsible for deciding when the start time of the interval was, we don't need to change that module. So we'll assume that we can leave it unchanged. Similarly, the data for the test that we've defined will work for the revised module, although it now only covers two intervals rather than three. With these decisions, we now know enough to be able to create a modified row for our traceability matrix. This is illustrated in Figure 8.

Requirement(s)	Specification(s)	Software Module(s)	Test(s)
R9.1.1a	S38.17a	DDM031, VSM022 v2, DSM040	VT063

Figure 8
Revised Traceability Matrix Row Example

We've just illustrated how you can use traceability as part of your configuration management process to control changes to elements of your system. We knew what elements were interrelated and could change all of them in a synchronized fashion.

While building a system, you want to keep the requirements for the system, the documentation for the system, and the physical implementation of the system synchronized. Of particular concern is keeping software items controlled. You can use configuration management software systems to control software items, making sure that a

completed and tested item is not modified without going through the process of reviewing and evaluating any proposed changes, getting approval from the configuration control board (or whatever change authority is established for the project). Software configuration items, because they exist only in electronic form,[14] can exist in electronic libraries managed by configuration management software that lets programmers "check out" a controlled software configuration item (program) to modify it. Programmers then "check in" the modified program after a test group validates that the modified program works correctly. The configuration management software keeps copies of all versions of a software configuration item. Thus, if a new version of a software configuration item turns out to be flawed, even after testing, you can replace it with the previous version, reverting to the prior system state. In the example we used, we changed the requirement for a software module, then replaced the first version of the software module with a second version, which addressed the revised requirement. If we'd needed to replace the module because of a bug, we'd still have indicated that there was a new version of the program. But we might have numbered it as "version 1.01" to show that the revised version didn't have changed functionality.

Although we didn't show this in our example, you can also correlate hardware items to other system elements as we did in our traceability matrix row examples. You could add columns to show which detectors (individually or by type) you used to count traffic volume and to show which computers you ran the software on. If you do this, the traceability matrix gets larger and, usually, more complex. That's why people use software systems and databases to handle configuration management. You can use these same systems, or similar ones, to manage your hardware configuration as well, as long as the systems have the capability to distinguish among different kinds of configuration items. During system development you track both which hardware configuration items you have installed (and where) and which you have successfully tested and integrated with the appropriate software. If the new system requires upgrades to existing hardware, you should use your configuration management system to track the upgrades as well.

So what happens after the system is implemented and in operation? That's what we cover in the next section.

Using Configuration Management During System Operation

Once a system is operational, it must keep working without unplanned interruptions. If something in the system breaks or wears out, you replace or repair it, depending on which is more efficient. What you don't want is a problem that brings your system down. Configuration management is a tool that can help you manage changes, minimizing system down time due to unexpected "glitches".[15] Whether the government organization operating the system is performing maintenance on all elements of the system or has contracted out for maintenance on any part of it, including software, the government agency is the organization responsible for overall maintenance of the system. Under

[14] The listing of a computer program is not really the software.
[15] A "glitch" is a software problem that causes a system failure.

these circumstances, the agency needs to have a plan for continuing to manage the system configuration.

Is configuration management still important on an operating system? Consider the following two cases.

Case 1: In June 2001, a software "glitch" prevented the New York Stock Exchange (NYSE) from trading stocks for almost 90 minutes.[16] The financial markets felt the impact even beyond the NYSE trading floor. Since investors couldn't calculate market indexes without NYSE data, trading also stopped at the American Stock Exchange and some futures and options markets. It also slowed trading on the NASDAQ Stock Market, due to investor reluctance to do business without information on NYSE trading. A new software installation caused the problem. The NYSE had installed the software on 8 of its 20 trading terminals and the system tested out the night before. However, the morning of June 8th, it failed to operate properly on the 8 installations. The NYSE tried to switch back to its old software, but was unable to do so before the opening of the trading session. Although you might see this as a failure of the NYSE's configuration management process, in reality, it was a success. Although the problem didn't arise until right before the opening of trading, the NYSE recovered from the problem relatively quickly. The computer system problems caused some red faces at the NYSE, but they minimized the damage. They were back up and operating within 90 minutes. Had the problem continued longer, the repercussions would have been more severe.

Case 2: ITS systems are just as vulnerable to "glitches," perhaps even more so. Take the example of an automated traffic signal control system with signals at 500 intersections, one or more timing plans at each intersection, and a central database that contains a copy of the timing plan(s) for each intersection. When technicians go to deal with a timing problem at specific intersections, a technician may make a change to one or more of the timing plans operating on a signal. For many reasons, the agency operating the automated traffic signal control system wants to know what timing plan(s) were in effect at each signal intersection and when each plan was in effect. What you hope is that the technician records the fact that he or she changed the timing plan(s), when, and to what.

If the technician fails to record the change he or she made, or if the change isn't recorded in the central database, the agency's configuration records are incorrect. If then a second technician has to service the same intersection, the second technician may restore the signal's timing plan(s) to values inappropriate for current conditions. After all, the second technician thinks that the correct timing plan(s) for the signal are what the central database has, not the changed values that weren't entered there.

Whether you see this as a failure of configuration management or as a problem with the first technician, the effect is the same. An important configuration change didn't get recorded. A subsequent change to the same configuration item put it into an undesired state. Good configuration management discipline would have prevented the error.

[16] *The Washington Post*, "Computer Glitches Hit NYSE," June 9, 2001, p. E01

Web Development, Content Management, and Configuration Management

Although the software development industry has used configuration management successfully for many years and most configuration management systems handle software development easily, the area where they don't work well is web development. If your ITS project involves web site development, you have some special things to think about. Configuration management works well for keeping track of the tools that you use to build the web site and any databases or scripts that you may use to build and manage your web site. However, it doesn't work well at all for managing the content of the web site, once it's been built and fielded.

Content management refers to the management of the information that you display or provide on your web site. If all of the information on all of the pages in your web site is *static* (i.e., it doesn't change), then you could probably use a configuration management system to manage each page. This would mean that each page, when it needed to be changed, would be "checked out" of the configuration management system, modified, reviewed, moved into production, and then the changed page would be "checked into" the configuration management system to await the next modification. You could do this with a configuration management system. But would you?

Probably not. Users concerned about how quickly they can get information out to their "customers" usually manage web content. These users aren't likely to be willing to adhere to the discipline of configuration management when all they change is the text of an informational page. And, in the fuzzy world of "content," it's hard to come up with a convincing argument for applying the discipline that we normally apply to managing software and hardware.

If we're dealing with *dynamic* content management, the issue is more complex. *Dynamic content management* usually means using an application that allows you to change the content of a web page depending on either who's reading it or on what actions the reader takes. For example, you may establish user profiles (or you may give a frequent user of your web site the ability to establish their own profiles) that you use to tailor the specific content of your web pages to the interests and needs of that particular user. There are many web creation and content management systems that allow you to do this today. However, you can forget about having any of these pages in a controlled, disciplined configuration management system. You may control the *templates* for your web pages in a configuration management system, but not the actual pages themselves.

However, using configuration management does make sense if you're *building* a new web site or making major *revisions* to an existing one. Let's take the new web site case first.

Building a new web site involves creating both content and structure. Each page contains either text or graphics and usually both. Pages may also have *scripts* associated with them that are used to change the content of the page, based either on reader interaction

(e.g., initiating a search of the site) or on pre-defined processes initiated by the site owner (e.g., web ads). Web site developers may also create *style sheets* that apply to some or all of the pages on a web site. Style sheets define the manner in which pages are formatted for display, enabling web developers to change the look of a web site by changing only the style sheet(s) instead of having to change each page individually. Since what's being done is similar to the development of a new product (you're building components and tying them all together), it makes sense to use a configuration management approach to controlling and releasing these components for use.

Similarly, if you're making major revisions to your web site, you want to hold off displaying those revisions until all related or linked pages have gone through the revision process. So you would probably put at least those pages that you wanted to change under configuration control, allowing you to make the changes in an organized fashion and them move the pages back into "production" once everything was ready. For example, if you were making changes to an existing web site to bring it into compliance with the requirements of section 508 of the Americans with Disabilities Act, you might pull all pages requiring re-work and control their re-issuance with a configuration management system. This would allow you to track your progress toward full compliance. The ability to track the progress toward completing a major site revision is a useful feature of configuration management.

Configuration Management Costs

In a perfect world, configuration management would be a tool we could employ knowing it would solve the problems that it is designed to address. We'd give using it no more thought than using a project schedule. But, just as using a project-scheduling tool to manage a project schedule involves some effort on our part, configuration management doesn't come free. There are costs associated with it.

One cost is the configuration management system itself. The number of configuration management systems in the market place changes frequently. It is impossible to have an accurate, up-to-date list all of the systems in the market in a document like this. However, the International Council on Systems Engineering (INCOSE) web page, http://www.incose.org/tools/tooltax/cm_tools.html, is one place to look for information on configuration management systems. Through its Tools Database Working Group, INCOSE tries to maintain current and accurate information on all systems engineering tools available to practitioners. Check that web site for information on products and vendors of configuration management systems.

Another important cost is the cost of administering the configuration management system itself. Although a contractor on an ITS project may handle configuration management for the project, the contractor passes on the cost of doing so to the public sector in the contractor's fees or labor charges. As a public sector manager, you need to plan for and incorporate these costs into your initial budget submission for your project. More importantly, you should consider what you'll do when the project is over and the ongoing configuration management responsibilities for the system become yours (or those of

someone in your organization). Configuration management doesn't stop just because the product has been developed. It's an ongoing need as the system evolves and requires maintenance. So you should look beyond the immediate project budget and consider how you'll integrate configuration management processes into your organizational structure.

While you're building the system, you can assign configuration management responsibilities to a contractor, either the software development contractor or an independent verification and validation contractor. Thus the government agency that contracted for the work doesn't have to expend its own people to manage the system configuration. If you plan to do this from the start of the project, you can budget in moneys for your configuration management effort. During the system development project, the configuration management team reports to the government project manager and to the project's configuration control board. Software development contractors are usually knowledgeable about configuration management and have people on their staff trained to perform the necessary functions. If the prime contractor for the system isn't a software development contractor but is used to building systems, the contractor usually has experienced, knowledgeable people who can perform the necessary configuration management for the project.

However, this situation changes once the system is implemented because, unless you keep a contractor around to do configuration management for you, the responsibility for keeping up the implemented configuration devolves to the government agency that acquired the system. If your agency is going to maintain the system, you must make clear from the outset of the project that you own the configuration management data and that your people must have access to and training on the configuration management system used during the project. This means that you must acquire a copy of the configuration management software used, if any, since the contractor isn't going to give it to you for free. But, although the system isn't necessarily yours as a project deliverable, the data in that system should be. Don't assume that you'll get the data automatically. Make getting it part of the contract between you and your contractor.

What If You Don't Use Configuration Management?

There's a commercial that you may have seen. An auto mechanic talks about a costly engine repair that could have been avoided if the car's owner had replaced his oil filter. The mechanic says, "You can pay me now, or you can pay me later." The quote's just as valid with regard to configuration management.

You can avoid the costs associated with configuration management by not bothering to employ it on your ITS projects. If you do, you'll probably pay instead in costs for:

- Figuring out which system components to change when requirements change

- Re-doing an implementation because you implemented to meet requirements that had changed and you didn't communicate that to all parties

- Losing productivity when you replace a component with a flawed new version and can't quickly revert to a working state

- Replacing the wrong component because you couldn't accurately determine which component needed replacing

Is that enough?

The reason that configuration management is included as a key systems engineering practice is simple. It works! It keeps you from incurring costs you need to incur. And, good systems engineers have learned, through practical experience, that it pays for itself many times over.

Don't pay the price later! Use configuration management.

Configuration Management Tools

Although it's possible to conduct a configuration management program with no more than a word processor and an electronic spreadsheet program on a personal computer, it's much easier to perform effective configuration management if you use a configuration management software tool. Commercially available configuration management tools are generally very useful, particularly on projects with large numbers of configuration items to control or with complex configuration management requirements. In particular, software-intensive projects (and ITS projects are frequently software-intensive) benefit greatly from the use of configuration management tools.

We prefer not to recommend specific configuration management tools, for a couple of reasons. First, the configuration management tool industry, like most software tool industries today, is a competitive one that can change dramatically in a short time as vendors upgrade their products. Any recommendation or ranking that we might make of current tools would probably change by the time you read this document. Second, any ranking or recommendation we might make may not apply to your project. Different tools have different strengths and weaknesses and you need to assess for yourself which ones best map to your needs.

To investigate configuration management tools, you can conduct a literature search, a market survey, or just use a search engine on the Internet. Searching the Internet with "configuration management" as your search argument will get you a fairly broad list of sites to investigate. They'll include both product sites and sites of configuration management practitioners. They'll also include sites that contain summarized information about configuration management products. Two sites that you'll probably find with an Internet search are those of the International Council on Systems Engineering and of the Institute of Configuration Management.

Let's discuss what you'll find on each site.

International Council on Systems Engineering (INCOSE)

INCOSE is an international non-for-profit organization, founded in 1990, with the stated goal of promoting systems engineering as a means of "enabling the realization of successful systems." Their web site address is **www.incose.org**. INCOSE has many resources of use to systems engineers and to those interested in systems engineering. Among those resources is their Tools Database, accessible from their home page.

The INCOSE Tools Database contains information that INCOSE has collected through the efforts of its Tools Database Working Group (TDWG), compiled in a series of projects. Among those projects is a classification of systems engineering tools by taxonomy. Although these taxonomies contain more than just configuration management tools, we're concentrating on the configuration management tool information that they provide. The three taxonomies offered by INCOSE are:

- **IMPIG (Information Model & Process Interest Group) taxonomy** – this is an INCOSE group that has established a categorization (taxonomy) for systems engineering processes. At its highest level, this taxonomy separates systems engineering tools into four classifications:

  - Management tools
  - Engineering tools
  - Information sharing tools
  - Infrastructure support tools

  You'll find configuration management tools under "management tools." Following the links in this category brings you to a table that lists the following information:

  - Tool name
  - Vendor
  - Description (of the tool)
  - Hardware supported
  - Operating systems
  - Universal resource locator (web locator for the product's vendor)
  - Last updated (date the information on the product was last updated by INCOSE)

  You can use the data in this table to find tools to investigate and consider for your project.

- **EIA-612 taxonomy** – EIA-612 refers to the general systems engineering standard, *Process for Systems Engineering*, originally developed by the Electronics Industry Association (EIA) and then adopted by the American National Standards Institute (ANSI) as the National systems engineering standard. In this taxonomy, you can find systems engineering tools in the "Control Process" category, under the major category, "Technical Management." Following the links yields a table with the same columns as under the IMPIG taxonomy. However, this table has different rows, since it contains more than just configuration management tools.

- **IEEE-1220 taxonomy** – The title of this taxonomy refers to the Institute of Electronic and Electrical Engineers (IEEE) general systems engineering standard, *Standard for the Application and Management of the Systems Engineering Process*. The IEEE, another National standards-making body, has developed its own set of standards for systems engineering and software engineering. In this taxonomy, you'll find configuration management tools under the "Control Process" category as well. Following these links gets you to the same table as does the IMPIG taxonomy.

INCOSE is the more general and more objective of the two major sites where you can find categorizations of configuration management tools. But let's look at what's at the second site.

Institute of Configuration Management (ICM)

The ICM bills itself as "the world's leading authority on configuration management and business process infrastructure." Its web site is **www.icmhq.com**. It was established in the 1970s and has developed a proprietary version of the configuration management process, which it labels "CMII." The ICM has a certification program for configuration management professionals who adopt the CMII approach. It certified its first configuration management professional in 1987 and has since certified over 2,500 others. Arizona State University and the University of Tennessee are co-sponsors of its certification program and offer course related to the certification process on their campuses.

CMII has achieved some success in industry and has been adopted as a configuration management standard by major companies in the aerospace, automobile manufacturing, and telecommunications industries.

The ICM has also certified several products as being CMII-compliant. You can find the list of these products in a table on their web site. The table lists the following information:

- System type (this refers to ICM categorizations and you will need to review their material to determine specifically what "type" means.)
- System name
- Release version
- Provider's name and web site
- Date certified

Since the ICM is only listing products that it has certified as being compliant with its CM approach, this list is much smaller than the one provided by INCOSE.

As we said before, we're not recommending any one approach or any one tool. We're simply providing information on how you can find out what tools are available and begin to decide which ones work best for you and your project.

Good luck!

References

ANSI/EIA Standard, ANSI/EIA-649-1998, *National Consensus Standard for Configuration Management,* Electronic Industries Alliance, Arlington, VA: 1998

Buckley, Fletcher J., *Implementing Configuration Management: Hardware, Software, Firmware, Second Edition,* IEEE Press, Piscataway, NJ: 1993

Daniels, M. A., *Principles of Configuration Management,* Advanced Application Consultants, Inc., Rockville, MD: 1985

Estublier, Jacky, (Ed.) *System Configuration Management,* 9th International Symposium, SCM-9, Conference Proceedings, Springer-Verlag, Berlin: 1999

Hajek, Victor G., *Configuration management: Describing how engineering and management disciplines can be applied to achieve the effective use of resources and the complete and accurate description of the end products,* Industrial and Commercial Techniques, Ltd., London: 1968

IEEE Standard, IEEE Std 828-1998, *IEEE Standard for Software Configuration Management Plans*, Institute of Electrical and Electronics Engineers, Inc., New York: 1998

Kelly, Marion V., *Configuration Management: The Changing Image,* McGraw-Hill Book Company, Berkshire, England: 1996

Lyon, David D., *Practical CM: Best Configuration Management Practices for the Twenty-First Century,* Butterworth-Heineman, Oxford: 2000

Magnusson, Boris (Ed.) *System Configuration Management,* ECOOP '98, SCM-8 Symposium Conference Proceedings, Springer-Verlag, Berlin: 1998

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

Smith, Brian L. and Bayne E. Smith, "An Evaluation of the Need for Configuration Management in Transportation Management Systems," The Transportation Research Board, November 2000

Software Program Managers Network, *The Condensed Guide to Software Acquisition Best Practices*, February, 1997

Software Program Managers Network, *Little Yellow Book of Software Management Questions*, February, 1997

Watts, Frank B., *Engineering Documentation Control Handbook: Configuration Management, Second Edition,* Noyes Publications, Park Ridge, NJ: 2000