SAE International

SURFACE VEHICLE/ AEROSPACE RECOMMENDED PRACTICE

SAE JA1005 MAY2012

Issued 2001-05
Reaffirmed 2012-05

Superseding JA1005 JAN2004

Software Supportability Program Implementation Guide

RATIONALE

JA1005 has been reaffirmed to comply with the SAE five-year review policy.

Foreword—This document identifies recommended practices for the implementation of a supportability program for software within an overall systems engineering framework. Guidelines for implementation of a Software Supportability Plan and associated Software Supportability Case are presented. Recommended practices are described for establishing a software supportability program through selection of life cycle activity tasks tailored for the application. Recommended models and process methods to achieve the life cycle activity tasks are briefly reviewed and/or referenced. The recommended practices are applicable to all projects incorporating software. The target audience for this document includes software acquisition organizations, logisticians, developers, supporters, and customers. This document is intended to be guidance for business purposes and should be applied when it provides a value-added basis for the business aspects of development, use, and sustainment of support-critical software.

TABLE OF CONTENTS

TO PLACE A DOCUMENT ORDER: **Tel: 877-606-7323 (inside USA and Canada)**
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
SAE WEB ADDRESS: http://www.sae.org

SAE values your input. To provide feedback on this Technical Report, please visit
http://www.sae.org/technical/standards/JA1005_201205

LIST OF ILLUSTRATIONS

1. Scope

1.1 Purpose—This SAE Recommended Practice provides recommended guidelines and best practices for implementing a supportability program to ensure that software is supportable throughout its life cycle. This Implementation Guide is the companion to the Software Supportability Program Standard, SAE JA1004, that describes, within a Plan-Case framework, what software supportability performance requirements are necessary.

This document has general applicability to all sectors of industry and commerce and to all types of equipment whose functionality is to some degree implemented via software. It is intended to be guidance for business purposes and should be applied when it provides a value-added basis for the business aspects of development, use, and sustainment of support-critical software. Applicability of specific recommended practices will depend on the support-significance of the software, application domain, and life cycle stage of the software.

1.2 Audience—The target audience for the document includes software acquisition organizations, developers, logisticians, supporters, and customers.

1.3 Applications—This document is applicable to all software-intensive projects, and in particular to projects where a long system software life cycle is expected. This document is applicable throughout the complete life cycle of software-intensive systems, although specific recommended approaches may be more effectively applied at specific life cycle points.

The combination of the Software Supportability Program Standard, SAE JA1004, Software Support Concept, SAE JA1006, and this document, provides guidance to implement software support-related aspects for:

a. System software logistics management as described in [DEFS0060-3], [MILHDBK502], [MILPRF49506], [MILHDBK347], [MILHDBK1467], [CECOM95]; and,
b. Software maintenance and supply processes and other related supporting processes described in [ISO12207], [ISO14764], [IEEE12207-0], [IEEE12207-1], [IEEE12207-2], [IEEE1219], and [SWEBOK].

2. References

2.1 Applicable Publications—The following publications form a part of this specification to the extent specified herein. Unless otherwise indicated, the latest version of SAE publications shall apply.

2.1.1 SAE PUBLICATIONS—SAE publications are available at: http://www.sae.org/

SAE AIR 5121—Software Supportability—An Overview
SAE ARP 5580—Recommended Best Practices for FMECA Procedures
SAE JA1000—Reliability Program Standard
SAE JA1002—Software Reliability Program Standard
SAE JA1004—Software Supportability Program Standard
SAE JA1006—Software Support Concept
SAE JA1010—Maintainability Program Standard

2.1.2 AFOTEC PAMPHLETS—Available from HQ Air Force Operational Test and Evaluation Center, http://www.afotec.af.mil

[AFOTECP2] AFOTEC Pamphlet 99-102, Volume 2, "Software Support Life Cycle Process Evaluation Guide," HQ Air Force Operational Test and Evaluation Center, August 1, 1994.
[AFOTECP3] AFOTEC Pamphlet 99-102, Volume 3, "Software Maintainability Evaluation Guide," HQ Air Force Operational Test and Evaluation Center, September 1, 1996.
[AFOTECP5] AFOTEC Pamphlet 99-102, Volume 5, "Software Support Resources Evaluation Guide," HQ Air Force Operational Test and Evaluation Center, August 28, 1995.

2.1.3 CECOM PUBLICATION—Available from http://www.monmouth.army.mil

[CECOM95] US Army CECOM, Potoczniak, J.J. "Software Logistics Planning Handbook," October 1995.

2.1.4 UK DEFENCE PUBLICATIONS—Available from http://www.dstan.mod.uk

[BS7000-5] British Standard 7000 (Part 5), "Design Management System. Guide to Management Obsolescence," 2001
[DEFS0040-1] UK Defence Standard 00-40(Part 1)/Issue 4(ARMP-2), "Reliability and Maintainability Part 1: Management Responsibilities and Requirements for Programmes and Plans," October 22, 1999
[DEFS0041] UK Defence Standard 00-41/Issue 3, "Reliability and Maintainability MOD Guide to Practices and Procedures," Section 15.1, June 25, 1993
[DEFS0042-2] UK Defence Standard 00-42 (Part 2)/Issue 1, "Reliability and Maintainability Assurance Guides Part 2: Software," September 1, 1997
[DEFS0042-3] UK Defence Standard 00-42 (Part 3)/Issue 1, "Reliability and Maintainability (R&M) Assurance Guidance Part 3: R&M Case," October 22, 1999
[DEFS0060-0] UK Defence Standard 00-60 (Part 0)/Issue 5, "Integrated Logistic Support Part 0: Application of Integrated Logistic Support," May 24, 2002
[DEFS0060-1] UK Defence Standard 00-60 (Part 1)/Issue 2, "Integrated Logistic Support Part 1: Logistic Support Analysis (LSA) and Logistic Support Analysis Record (LSAR)," March 31, 1998
[DEFS0060-2] UK Defence Standard 00-60 (Part 2)/Issue 5, "Integrated Logistic Support Part 2: Logistic Support Analysis Application to Software Aspects of Systems," May 24, 2002
[DEFS0060-3] UK Defence Standard 00-60 (Part 3)/Issue 2, "Integrated Logistic Support Part 3: Guidance for Application Software Support," March 31, 1998

2.1.5 IEC PUBLICATION—Available from International Electrotechnical Commission, 3, rue de Verambe, P.O. Box 131, 1211 Geneva 20, Switzerland.

[IEC 60812] IEC 60812:1985, "Analysis Techniques for System Reliability - Procedure for Failure Mode and Effects Analysis (FMEA)," 1985

2.1.6 IEEE PUBLICATIONS—Available from IEEE, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

[IEEE1219], IEEE 1219, "IEEE Standard for Software Maintenance," June, 1998
[IEEE12207-0] IEEE 12207-0-1996, "Software life cycle processes," IEEE Computer Society, March 1998
[IEEE12207-1] IEEE 12207-1-1997, "Software life cycle processes—Life cycle data," IEEE Computer Society, April 1998
[IEEE12207-2] IEEE 12207-2-1997, "Software life cycle processes—Implementation considerations," IEEE Computer Society, April 1998

2.1.7 ISO PUBLICATIONS—Available from ANSI, 11 West 42nd Street, New York, NY 10036-8002.

[ISO12207] ISO/IEC 12207, "Information Technology - Software Life Cycle Processes," August 1, 1995.
[ISO14764] ISO 14764, "Information Technology – Software Maintenance," ISO, 1999.
[ISO15504] ISO/IEC TR 15504-1:1998 Information Technology -- Software Process Assessment," Parts 1-9, ISO JTC 1/SC 7, August-September 1998.

2.1.8 MILITARY PUBLICATIONS—DODSSP, Building 4/Section D, 700 Robbins Avenue, Philadelphia, PA 19111-5098, http://dodssp.daps.mil/

[MILHDBK347] MIL-HDBK-347, "Mission-Critical Computer Resources Software Support," May 22, 1990
[MILHDBK502] MIL-HDBK-502, "DOD Handbook Acquisition Logistics," May 30, 1997
[MILHDBK1467] MIL-HDBK-1467, "Acquisition of Software Environments and Support Software," November 19, 2002
[MILHDBK2155] MIL-HDBK-2155, "Failure Reporting, Analysis and Corrective Action System," December 11, 1995
[MILPRF49506] MIL-PRF-49506, "Performance Specification Logistics Management Information," November 11, 1996

2.1.9 SEPO PUBLICATION—Available from Software Engineering Process Office, D12, Space and Naval Warfare Systems Center San Diego, 53560 Hull Street, San Diego, CA 92152-5001, http://sepo.spawar.navy.mil

[SEPO] Software Engineering Process Office (SEPO), Home page focal point for the Space and Naval Warfare Systems Center.
[SEP02] "Software Estimation Process," Software Engineering Process office (SEPO), version 2.2 August 1999

2.1.10 USC PUBLICATION—Available at: http://sunset.usc.edu/COCOMOII/cocomo.html

[USC] University of Southern California, COCOMO Project Suite Home Page

2.1.11 OTHER PUBLICATIONS

[BOEHM81] Boehm, V.W., "Software Engineering Economics," Prentice-Hall, Englewood Cliffs, NJ, 1981
[BS5760] British Standard, BS5760: Part 5:1991, Reliability of systems, equipment and components: Guide to failure modes, effects and criticality analysis (FMEA and FMECA), British Standards Institution, London, England, 1991
[CASA-SAS] CONSTRUCCIONES AERONÁUTICAS, S.A.(C.A.S.A.), "Support Analysis for Software Processes: Generic Process Flow," Integrated Logistic Support, Draft D, Madrid, Spain, April 14, 2000

[HUTCH00] Hutchison, P., "Using Def Stan 00-60 LSA Reports to Address Software Supportability," MSc Dissertation in Software Engineering, Oxford University, March 9, 2000

[LMEPI140-xx] EPI 140-xx (Draft 6), "Software Reliability and Maintainability Best Practices," Lockheed Martin Corporation, EPI Center, Camden, NJ, August 1999

[LYU96] Lyu, Michael, Handbook of Software Reliability Engineering, McGraw Hill / IEEE CS Press, Washington, DC, 1996

[MOOD98] Harrison, R., Counsell, S., Nithi, R., "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," IEEE Transactions on Software Engineering, Washington, DC, Vol 24, No 6, 1998, pp 491-495

[MUSA92] Musa, John D. "Operational Profiles in Software Reliability Engineering," IEEE Software, Washington, DC, March 1993, pages 14-32

[MUSA99] Musa, J., Software Reliability Engineering, McGraw-Hill Book Company, NY, 1999

[NATO96] NATO (Draft), "COTS Software Acquisition Guidelines and COTS Policy Issues – 1st Revision," NATO Communications and Information Systems Agency, Brussels, Belgium, January 12, 1996

[NATO97] NATO (Draft), "NATO Guidelines for the Integration of Off-The-Shelf Software," Working Paper AC/322(SC/5)WP/4, NATO C3 Board Information Systems Sub-Committee, Brussels, Belgium, June 30, 1997

[PEERCY1] Peercy D., Tomlin,E., Horlbeck, G., "Assessing Software Supportability Risk: A Minitutorial," IEEE Computer Society, Proceedings of the Conference on Software Maintenance, 1987, Washington, DC, September 21-24, 1987

[PEERCY2] Peercy D. E., "A Software Maintainability Evaluation Methodology," IEEE Transactions on Software Engineering, Vol 7, No 4, 1981, pp 343-352, Washington, DC

[RCTADO178B] RCTA/DO 178B, "Software Considerations in Airborne Systems and Equipment," Federal Aviation Administration software standard, RTCA Inc., Washington, DC, December 1992; Notice N 8110.77, "Guidelines for the Approval of Field-Loadable Software," November 18, 1998; Notice N 8110.79, "Guidelines for the Approval of Field-Loadable Software by Finding Identicality Through the Parts Manufacture Approval Process," February 22, 1999; Notice N 8110.84, "Guidelines for the Approval of Airborne Systems and Equipment Containing User-Modifiable Software," April 8, 1999

[ROYCE98] Royce, W., Software Project Management – A Unified Framework, Addison-Wesley, 1998.

[SCHN98] Schneidewind, Norman, "How to Evaluate Legacy System Maintenance," IEEE Software, Washington, DC, July/August 1998, pp 34-42

[SCHN99] Schneidewind, Norman, "Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics," IEEE Transactions on Software Engineering, Vol 25, No 6, Washington, DC, November/December 1999, pp 769-781

[SEICBS] Software Engineering Institute, COTS-Based Initiative, " Workshop on COTS-Based Systems," CMU/SEI-97-SR-019, Pittsburgh, PA, November 1997

[SEICMM] Paulk, M., Curtis, B., Chrissis, M., Weber, C., "Capability Maturity Model for Software, Version 1.1," CMU/SEI-93-TR-024, Software Engineering Institute, Pittsburgh, PA, February 1993

[SWEBOK] Abran, A., Moore, J., Bourque, P., Dupuis, R., "Guide to the Software Engineering Body of Knowledge: Chapter 6 Software Maintenance," A Stone Man Version 0.7, Software Engineering Coordinating Committee of IEEE and ACM, Washington, DC, April 2000

[WEID98] Weidenhaupt, K., Pohl K., Jarke, M., Haumer, P., "Scenarios in System Development: Current Practice," IEEE Software, Washington, DC, March/April 1998, pp 34-45

3. Definitions—See Appendix A.

4. Overview of Software Supportability Program—Software supportability cannot be improvised; it must be addressed consistently and systematically. Supportability must be addressed throughout the complete life cycle from concept through development, delivery, and operational use, in a way that allows demonstration to the customer that support requirements are being met effectively. Addressing supportability early in the life cycle can avoid the possibility of costly product redesign by ensuring that supportability is designed into the product. The purpose of this section is to outline a broad approach that will enable the satisfaction of this objective within the context of the Software Supportability Program Standard [SAE JA1004]. The broad approach to ensure software supportability, based on a supportability analysis, is illustrated in Figure 1.

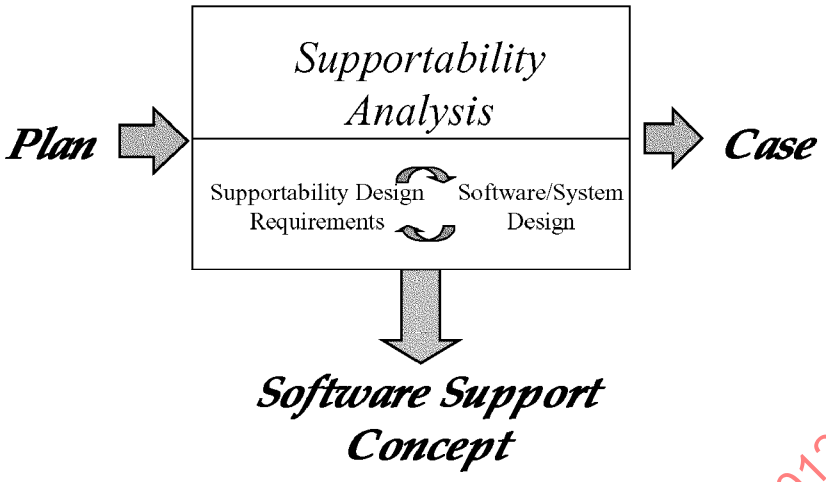


FIGURE 1—OVERVIEW OF SOFTWARE SUPPORTABILITY PROGRAM APPROACH

A software supportability plan defines the supportability analysis needed to drive out a software support concept. The supportability analysis results and logical arguments (analysis evidence) are captured in a software supportability case. Supportability analysis comprises methods and techniques that are typically undertaken as a tailored set of tasks. Supportability analysis also feeds supportability design requirements, which in turn, via the resulting software/system design, feeds the supportability analysis. The supportability design requirements form part of the software support concept output.

The standard Plan and Case are described in [SAE JA1004] and a software support concept framework is described in [SAE JA1006]. Suggested templates for the Plan and Case are given in Appendix B. Detailed information on the incorporation of analysis activities within the life cycle is provided in Section 5 and Appendix C. Best practice methods and techniques that facilitate supportability analysis are described in Section 6.

4.1 The Importance of Planning—Achieving the optimum cost, responsiveness, and quality of support for any particular software product will come from having an effective planning strategy directed towards supportability needs as determined through customer requirements. The strategy should be developed and executed in conjunction with planning for overall system supportability. Furthermore, there should be visibility to project and higher program managers of major milestones and outputs, to ensure that control and monitoring of supportability activity is properly integrated with other disciplines.

A software supportability plan, as described in [SAE JA1004], is the primary enabler for satisfying customer supportability requirements for a software product. In common with other types of project plans, a software supportability plan needs to address the software aspects of a system supportability plan, and describe the activities that are to be undertaken to achieve the software supportability objectives. The Plan also describes activities that are to be undertaken to demonstrate that software supportability objectives have been achieved. The Plan's activities, comprising associated tasks, schedules, resources and outputs, should be carefully integrated with appropriate system-level support planning activities.

Software supportability analysis, driven by a software supportability plan, should produce a software supportability case and software support concept. Where software development is carried out, supportability analysis evidence, detailed in a software supportability case, should include a set of design requirements for software supportability that are iterated with the system and software design effort to ensure supportability characteristics are built into the software products.

4.2 Determining Support Requirements Through Analysis—Software supportability analysis tasks are conducted throughout the life cycle to identify and complete relevant aspects of the software support concept and provide evidence, documented in a software supportability case, of supportability achievement. These tasks ensure that the customer requirements for supportability are clearly understood, are being met through the various management and engineering activities, and can be demonstrated to the customer's satisfaction.

Software supportability analysis tasks must, throughout the system life cycle, be conducted within the overall context of system and logistic engineering to ensure that a system support solution, and not just a software support solution, is attained. Once early supportability planning has been conducted, and progress review mechanisms defined, specific supportability analysis tasks are conducted in accordance with the software supportability plan. These analysis tasks can be described as addressing the following attributes of software support:

a. **Definition**—Establish software supportability objectives; determine related software support goals, thresholds, constraints, and performance; analyze software support cost and operational readiness drivers within the system context; and identify the general approach to software support in comparison with existing systems.
b. **Alternatives**—Determine optimal alternatives for the who, where, when, and how of software support to balance software and system life cycle cost, schedule, performance, and supportability.
c. **Resources**—Identify software support personnel, systems, facilities, training, and provisioning necessary to establish the selected transition and operational support alternative.
d. **Assessment**—Provide evidence throughout the life cycle that the supportability requirements are being or have been achieved and any prior deficiencies have been corrected.

A general framework and sequencing for software supportability analysis tasks is illustrated in Figure 2.

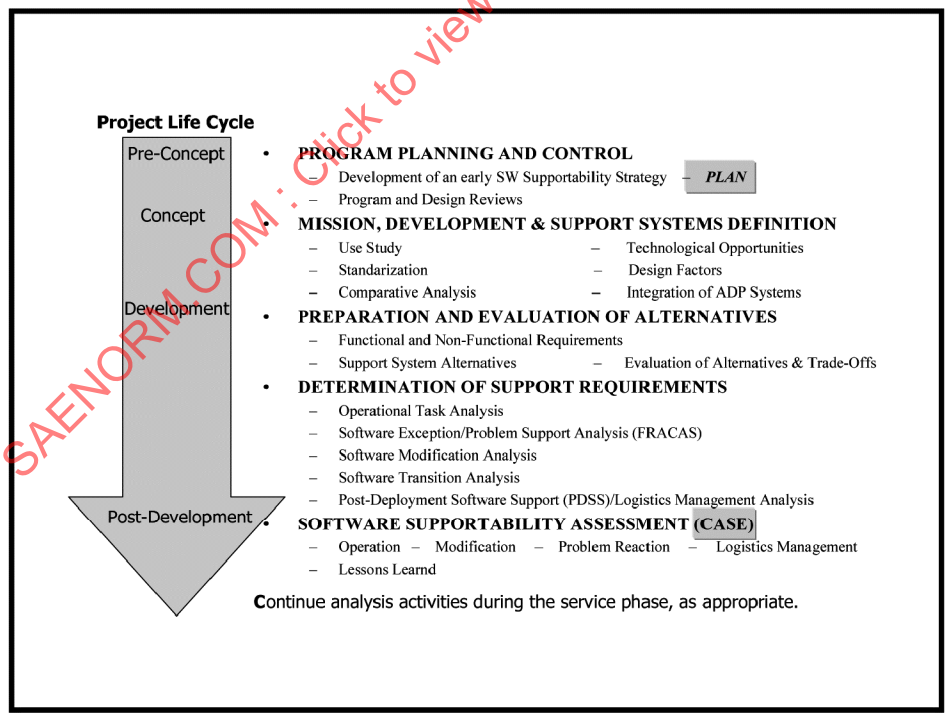


FIGURE 2—SOFTWARE SUPPORTABILITY ANALYSIS TASKS

Specific sequencing of tasks is described in Section 5 while the specific tasks are described in more detail in Appendix C. Best practice methods and techniques for achieving some of the tasks are given in Section 6. These analysis tasks are similar to those described as Software Logistics Support Analysis (LSA) tasks in [DEFS0060-3] which also includes thorough descriptions of generic software support processes that can form the basis for software support function analysis. A process flow for supportability analysis tasks can be found in [CASA-SAS].

4.3 Outputs of Supportability Analysis—Software supportability analysis, as defined by this document, has 3 outputs (reference Figure 1): a software support concept, supportability design requirements, and a software supportability case.

The software support concept and the software supportability case can be considered external outputs, in the sense that they are provided to the customer, even though they are used internally during the development program. The supportability design requirements, however, are to be considered internal outputs from the supportability analysis to the design area, and hence may not be seen by an external customer.

4.3.1 SOFTWARE SUPPORT CONCEPT—A software support concept is a structured description of the characteristics of a software product, and of the associated support infrastructure and processes, which are necessary in order to satisfy the customer's supportability requirements. A detailed generic description of a software support concept framework is presented in [SAE JA1006], and illustrated in Figure 3.

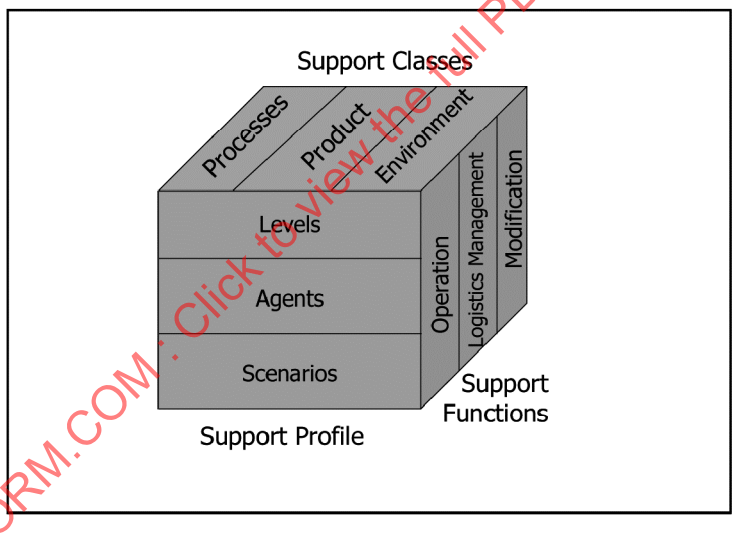


FIGURE 3—SOFTWARE SUPPORT CONCEPT FRAMEWORK

Appendix A gives definitions for 'support concept' and its three dimensions of 'support classes,' 'support functions,' and 'support profile.'

A software support concept for a particular product will evolve as the software is developed and more becomes known about the support requirements. The content of a Concept should be compiled from supportability analysis activities such as described in this guide. A Concept would normally be developed jointly with the customer, and finalized at, or prior to, acceptance.

Normally, it is essential to maintain a software support concept document throughout the operational life of the subject software. A Concept will describe the support facilities, resources and procedures that are actually fielded, and hence it is important that this information should be accurate and kept up-to-date.

4.3.2 SUPPORTABILITY DESIGN REQUIREMENTS—Design requirements for supportability are outputs from particular supportability analysis tasks described within this guide. Design requirements are concerned with ensuring the software is designed for supportability to facilitate possible future system and/or software changes. Such changes could include reuse or modification of the software for new applications. The desired supportability characteristics define the 'support classes' dimension of a software support concept. Confirmation that supportability characteristics have been achieved is captured in a software support case through evidence and logical argument.

Supportability analysis is conducted to determine the precise characteristics of each support class that are appropriate for a specific application. The goals of this activity are to deliver a supportable product, ensure that the necessary support infrastructure is in place before the entry into service, and sustain the software support during the software operational life. The support concept captures the support class characteristics that have been specified/achieved for a specific application [SAE JA1006].

The actual methods and techniques to be applied to a particular product to derive supportability design requirements should be agreed jointly between customer and supplier at the outset of the project and then reviewed throughout the project. The mechanism and format in which supportability design requirements are communicated between supportability analysts and software designers is mainly a supplier concern, but principal activities and milestones should be visible in a software supportability plan.

4.3.3 SOFTWARE SUPPORTABILITY CASE—A software supportability case has two main purposes: to provide a justification of the approach taken by the supplier to software supportability; and to document evidence, and argument, that demonstrates that the software meets (or exceeds) its supportability requirements. The general rationale and components of a software supportability case are described in [SAE JA1004]. An example outline for a software supportability case is provided in Appendix B. Other descriptions and uses of the Case construct can be found in [DEFS0042-2] and [DEFS0042-3].

A software supportability case will be progressively populated as a project proceeds through development, and/or construction, and/or postproduction support. The required scope and content of a software supportability case should have been determined through early decisions about the nature of the software products to be delivered to the customer, and the customer's needs for support. A software supportability plan should show how source data and associated information required for the Case would be provided.

A software supportability case would typically be a deliverable to the customer to support system acceptance, although for longer or more complex development programs there may be benefit in producing intermediate software supportability cases at appropriate milestones. Where, after initial delivery, a software product is expected to continue to grow and be enhanced, its software supportability case should be maintained and periodically reviewed.

5. Supportability Program Life Cycle Management—For the purposes of this guide, software is considered to be the logical design of a system that has the ability to be readily adapted to meet changing requirements, i.e., software provides the logical functionality of a system. The intention is for software faults to be corrected, enhancements incorporated, and changes made to adapt to environment modifications and alternate uses. Ideally, software should be designed with supportability characteristics built-in. The infrastructure of people, support systems, facilities, and processes are intended to facilitate an evolving software capability. However, this implies that the software supportability program is also intended to evolve throughout all life cycle phases. As such, the software supportability plan, supportability analysis, supportability case and support concept must be applicable throughout all life cycle phases and updated as appropriate. This life cycle management approach is illustrated in Figure 4 for a full-scale development project. Considerations for off-the-shelf guidance are provided in Appendix E.

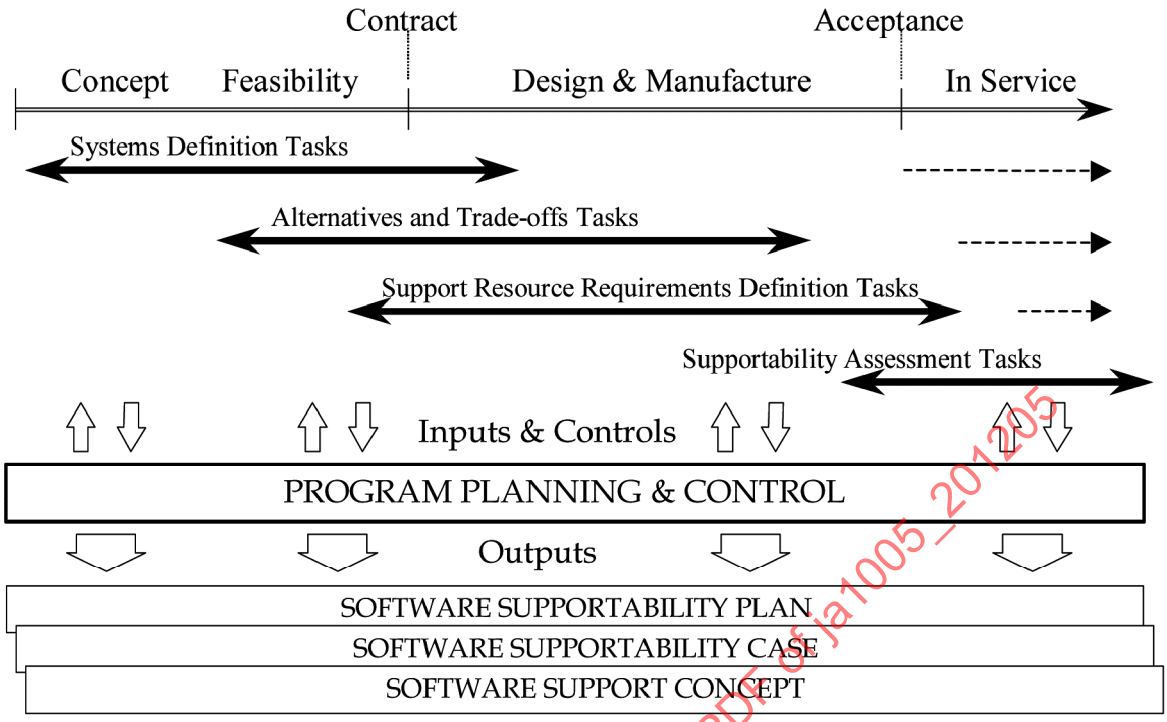


FIGURE 4—SOFTWARE SUPPORTABILITY LIFE CYCLE MANAGEMENT APPROACH

5.1 Concept and Feasibility Analysis—During the concept and feasibility analysis, the key element of the supportability program is the identification of customer supportability requirements. These supportability requirements should, as much as possible, be performance based and be related directly to system operational needs and cost constraints. Possible supportability performance requirements include:

a. Measure of supportability characteristics that have been built into the software product, support processes, and support environment design;
b. Maturity level, using for example [SEICMM], of software support supplier and associated support processes;
c. Software product planned release frequency, change request profile, and change response time for both emergency and normal software changes;
d. Software reliability metrics and design for reliability metrics that would affect corrective maintenance activity and confidence in the correctness of changes;
e. Support cost stratified by unit time, support functional area, supplier;
f. Software supportability risk: probability and impact of not performing a software support action in the time period required.

Supportability performance requirements must be quantifiable and measurable. In many cases the customer may only have a vague idea of what software support is required and, in particular, what the software supportability performance requirements should be. Perhaps such an application has never been done before, or, perhaps, there are other alternatives that due to cost, business or technological constraints are not easy to understand. Or, perhaps the customer has no clear vision and/or policy on support. In some cases, there is no specific customer, especially if the software product is not a custom developed product but is intended to be marketed as an Off-The-Shelf (OTS) element.

Determining the customer's software support requirements may be complicated, and the customer may require some help in the definition of these requirements. For OTS procurement, even somebody representing the customer, such as the marketing department, may require assistance in defining the support requirements.

Three steps should be followed during the concept and feasibility analysis phase:

a. Establishment of the support objectives;
b. Comparison of provisional supportability requirements with other existing systems and consequent refinement of supportability requirements as appropriate; and
c. Agreement on requirements and basic support concept framework.

Possible analysis tasks to address these three steps are illustrated in Figure 5. See Appendix C for the task descriptions.

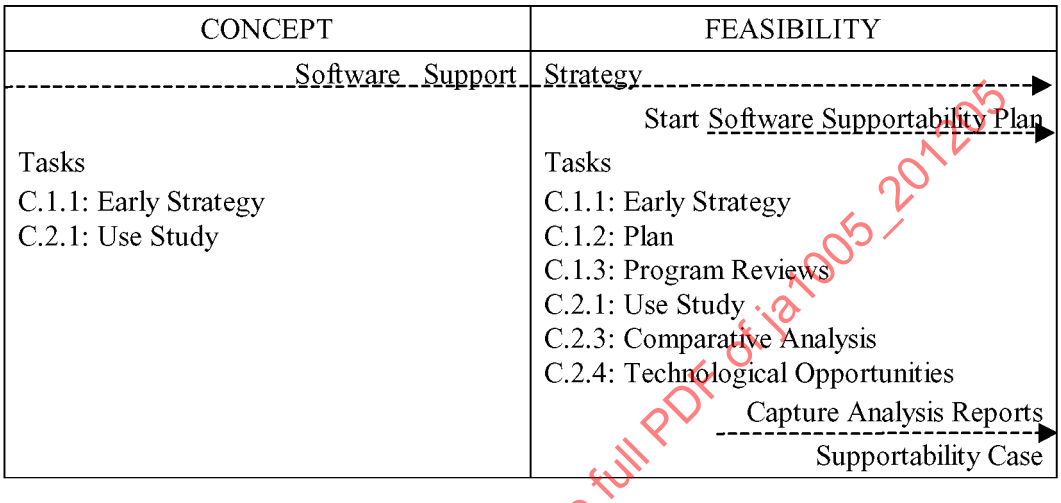


FIGURE 5—EXAMPLE CONCEPT/FEASIBILITY PHASE ANALYSIS TASKS

5.1.1 DEFINITION OF OBJECTIVES—The first aspect to be addressed is a clear definition of what objectives are sought and/or needed by the software supportability program. This statement of the objectives usually permits a simplification and clarification of the requirements. Objectives may include cost guidelines, rapid turn-round for new releases, limited interruption of user operations, use of OTS software components, response time to failures, and use of contractor and vendor support resources. It is important to clearly state key objectives and indicate their relative importance in case compromises are necessary.

5.1.2 CLARIFICATION OF TERMINOLOGY—The reference [SAE JA1006] provides a framework for development of a software support concept. However, it is useful to adopt application-specific terminology in order to ensure that all participants (particularly the customer) can more readily understand and communicate the concept. It is recommended that the terminology used in [SAE JA1006] be carefully reviewed and mapped to application-specific terms. For example, terms used for "Support Levels" may vary greatly for different industries, and even within different products of a same industry. A mapping of possible terminology for a few product categories against the terms used in [SAE JA1006] is illustrated in Figure 6.

The mapping and functions at each level may also change over time as technology changes. For example, in the civilian automotive industry, the organization/field (i.e., driver/owner) may not have any software support functions at this time. Owners do perform some hardware support functions. As software becomes a more pervasive and critical component in automotive subsystems, and network communication interfaces link end-users to direct logistic suppliers, the driver/owner may have software support actions that can be performed. Updates to software can be downloaded directly by the owner from the logistic supplier or embedded software components (updates or replacements) can be replaced by the owner. A remote site may run diagnostic traces with the assistance of the owner who attaches a diagnostic adapter kit that can be directly controlled. The key is for the support concept to capture the desirable approach and appropriately evolve over the life of the software product.

SAE JA1006	Civilian Automotive	Military Aircraft	Civilian Aircraft	Unmanned Space Vehicle	General computing
Organization / Field	Garage/In-Drive (Owner/Driver)	Flight Line / Field	In-Flight (Pilot)	Space	User
Intermediate	Dealer	Hangar	Airline	Manned Space Station / Vehicle	System Administrator
Depot	Vehicle Manufacturer	Military Facility / Depot	Airframe Manufacturer	Ground Control Center	Computer Center
Vendor	Supplier	Contractor / Vendor / Supplier	Equipment Supplier	Main Contractor / Supplier	Supplier / Subcontractor

FIGURE 6—EXAMPLE TERMINOLOGY ALLOCATION FOR DIFFERENT INDUSTRY AREAS AND PRODUCTS

5.1.3 DEFINITION AND TAILORING OF SUPPORT CONCEPT FRAMEWORK—A skeleton support concept may be derived from [SAE JA1006], even if it is only at the top level, as a kind of checklist in order to ensure that no support aspect has been forgotten. Once the terminology has been clarified, it is often quite easy to determine that certain aspects of the general support concept framework are not applicable. This tailoring of the framework for a specific application facilitates the planning and prioritization of supportability analysis tasks.

5.1.4 DEFINITION OF INITIAL REQUIREMENTS—An initial set of requirements is defined on the basis of outputs from the analysis tasks such as illustrated in Figure 5:

a. Objectives, requirements, program organization, system support integration;
b. Plans for Software Support Concept Profiles/Functions/Classes analysis tasks;
c. Use Studies, comparative analyses, functional requirements from existing systems;
d. Integration planning with product development/support engineering;
e. Outline of roles and responsibilities of involved support agents; and
f. Initial Software Support Concept with high-level support scenarios as output.

5.2 Design and Production—During the design and production of system and software components, a key element of any supportability program is meeting and demonstrating the customer supportability requirements. Refinement of the requirements is also typically accomplished as the software and its support concept become well understood within the system's operational purpose. The supportability analysis tasks should establish the best support concept definition and acquisition approach that balances supportability performance requirements, life cycle schedule constraints, and life cycle cost estimates.

Alternative support options, and tradeoff of supportability analysis tasks, initiated in the previous phase will be expanded in scope for the identified software support significant items to determine an optimum support concept: what support functions will be performed, by whom, and at what location(s) for the key scenarios required for system operational support. Specific support resource requirements for personnel, support systems, facilities, and processes will be defined so that a support capability can be transitioned/acquired to achieve the required supportability performance.

Possible analysis tasks to address this phase are illustrated in Figure 7. See Appendix C for the task descriptions.

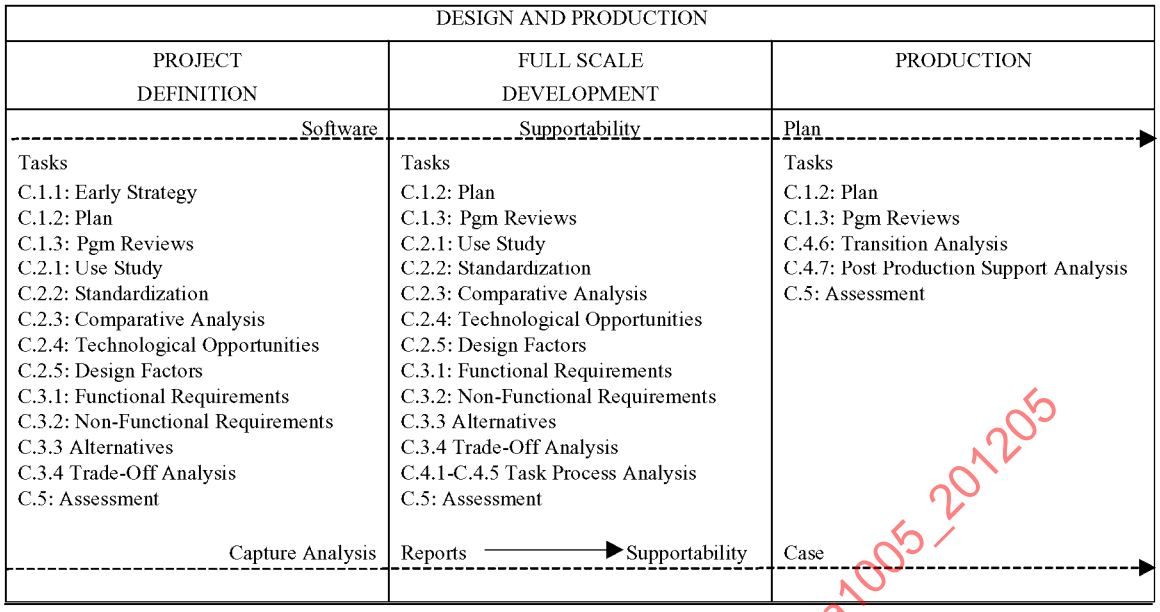


FIGURE 7—EXAMPLE DESIGN AND PRODUCTION PHASE ANALYSIS TASKS

The following support areas may be addressed to meet and demonstrate that customer requirements have been met during the design and production phase:

a. Design to class attributes, software support scenarios, support significant items;
b. Determination of support levels and characteristics;
c. Use of methods, models, and analysis processes;
d. Alternative support systems, trade-off analyses;
e. Integration with product development/support engineering activities;
f. Roles and responsibilities of support agents, definition of supplier contractual elements;
g. Final Software Support Concept;
h. Qualitative and quantitative supportability performance measurement;
i. Evidence collection process and presentation process;
j. Transition and fielding prototypes, support assessments;
k. Integration with product acceptance, delivery, operation, and support;
l. Contractual arrangements; and
m. In-service demonstration of capability.

5.3 In-Service—During the in-Service phase, software supportability documents (plan, concept, case) are evolved in conjunction with the software products being supported. The initial task is to conduct an as-acquired assessment. As the software support is evolved, nearly all analysis tasks identified in 5.1 and 5.2 could be repeated. The analyses will tend to be more localized around the implemented support concept dimensions. In general, all software supportability tasks may be applicable throughout the In-Service phase.

For example, a new technology may be developed and its potential impact on the system and its software supportability must be analyzed. On the other side, obsolesce may become a serious issue. Ongoing supportability analysis is accomplished using previous analysis reports, new technological opportunity analyses, and appropriate dependency analysis tasks relative to the appropriate support class, functional area, and/or profile. The Plan, Concept, and Case are appropriately updated to reflect any changes.

5.4 Tailoring Supportability Life Cycle Management—The tailoring of software supportability program activities (supportability analysis tasks) depends upon the following general factors:

a. Phase of the project when the program is established;
b. Type of the software (e.g., development item, Off-The-Shelf item, combination);
c. Amount of design freedom to affect software and its system interfaces;
d. Availability of resources and information on existing systems;
e. System engineering and integrated logistic support interfaces;
f. In-service information requirements for support agency and customer(s); availability of supportability analysis data; and
g. Applicability of activities to the task.

Tailoring tasks is a primary focus of the early support strategy and life cycle support planning. Some specific guidance on task dependencies, task tailoring logic, and system level task interfaces is provided in Appendix C. Other specific off-the-shelf task tailoring guidance is provided in Appendix E.

5.5 Relationship of Software Supportability Program to Existing Standards

5.5.1 [ISO12207] AND [IEEE12207-0] STANDARDS—These standards define five primary software life cycle processes (Acquisition, Supply, Development, Operation and Maintenance), a set of eight supporting processes (Documentation, Configuration Management, Quality Assurance, Verification, Validation, Joint Review, Audit, and Problem Resolution), and a set of four organizational processes (Management, Infrastructure, Improvement, Training). The joint use of [SAE JA1004], [SAE JA1006], and this implementation guide provides detailed guidance on managing and implementing the software support/ modification aspects covered by these ISO/IEEE standards. In particular, the Acquisition, Development, and Operation processes identify the interfaces and relationships of the production system to its software support processes. The Supply and Maintenance processes represent primary software support processes that overlap with the software support functional areas of Modification and Logistics Management. Configuration Management and Problem Resolution are directly tied to the software support activities and the other ISO/ IEEE support processes provide general software project assistance. The Plan-Case mechanism from [SAE JA1004] provides a natural link to the four organizational processes. As the software supportability program defines the requirements for supportability management, infrastructure, improvement and training, an initial part of the organizational process life cycle activities is established.

5.5.2 [DEFS0060-2] AND [DEFS0060-3] STANDARDS—These standards provide general system and software specific Logistic Support Analysis requirements and guidelines. The software supportability analysis tasks described in this document are consistent with [DEFS0060-3]. The general set of software support processes described in [DEFS0060-3] is illustrated in Figure 8 and can be interpreted in terms of the Plan-Case mechanism of [SAE JA1004], the software support concept framework described in [SAE JA1006], and the software supportability program and task analysis described in this document.

5.5.3 [MILPRF49506], [MILHDBK502], [MILHDBK1467] STANDARDS—These standards and guidebooks provide general system integrated logistic management performance requirements and implementation guidance for application to United States Department of Defense systems. In addition, [MILHDBK1467] provides detailed guidance for software support environment characteristics. Although primarily focused at the system level, these standards provide a capability to include hardware and/or software support concerns. [SAE JA1004], [SAE JA1006], and this document provide appropriate guidance to implement software support within the context of the referenced standards.

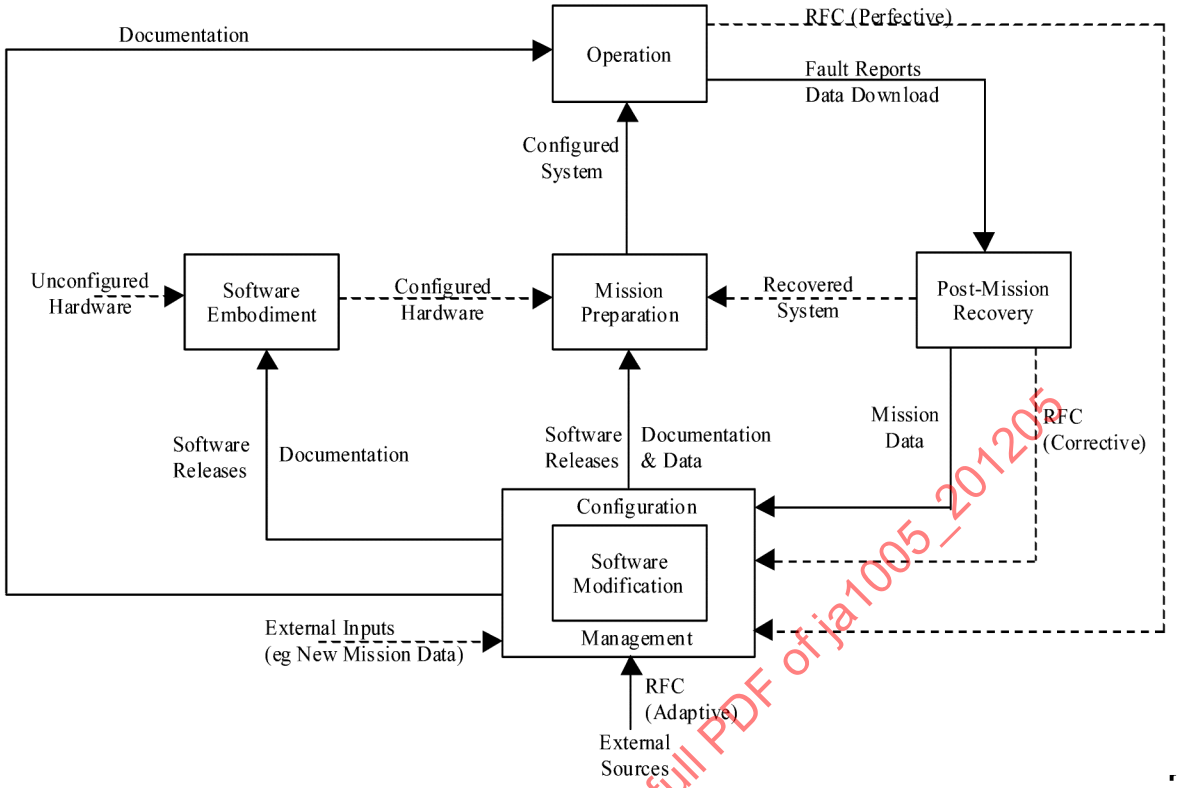


FIGURE 8—MODEL OF SOFTWARE SUPPORT PROCESSES [DEFS0060-3]

5.5.4 [SEICMM] AND [ISO15504] STANDARDS—These de-facto standards provide thorough mechanisms for understanding characteristics of desired software engineering and support-related processes and measuring an organization's maturity relative to those processes. Although these standards tend to focus primarily on development terminology, many of the key process areas described will affect the supportability characteristics of developed software and the capability of a supplier to provide software support services. Thus, the maturity performance level of an organization could be one part of a supplier's supportability process performance measure.

5.5.5 [SWEBOK] GUIDELINE—Although not a standard, this extensive set of software engineering guideline information contains a comprehensive chapter 6 on software maintenance. The software maintenance terminology used is closer in to the scope of software support and supportability. The [SWEBOK] reference provides an excellent overview of software maintenance issues and problems, processes, measurement, methods and techniques, relationships among the various standards as regards software maintenance, and an extensive list of references. The [SWEBOK] reference provides further details that would facilitate the implementation of a software supportability program in accordance with this Implementation Guide.

6. ***Supportability Analysis - Methods and Techniques***—There are many possible methods and techniques that can be used to facilitate the accomplishment of specific supportability analysis throughout the software life cycle phases of development, test, operation, and support. This section provides guidance on the following best practice methods and techniques for software supportability analysis:

   a. Use Study;
   b. Software Identification and Breakdown;
   c. Categorization and Selection of Support Significant Items;
   d. Support Scenario Modeling;
   e. Block Release Profile Modeling;
   f. Support Cost Estimation;
   g. Supportability Assessment; and
   h. Inter-discipline Methods and Techniques, e.g., Level of Repair Analysis, Failure Modes, Effect and Criticality Analysis, and Failure Reporting and Corrective Action System.

6.1 Use Study—The Use Study establishes a baseline for all support analysis. Such a Study outlines the intended operational use of the software, possible existing support systems that may be used for comparison or for undertaking the support, key support performance parameters (e.g., number of users, support response times, support scenario sequences), and any limitations that the Customer wishes to impose on the support concept (e.g., maximum cost, available staffing or expertise, strategic business decisions). In this context, it is important to ensure that the Customer and Supplier are both clearly aware of the Use Study purpose and that this Study requires some preparation and review as a team in order to ensure that there is a clear comprehension of all the software support aspects to be considered.

The Use Study is indeed an analysis activity – because it states the global framework that the Customer is considering for the support concept, and this usually requires a lot of definition work. The Use Study is a mechanism for determining and refining Customer requirements. While certain areas might be left open, awaiting further analysis, the Customer can usually state what support options are totally unacceptable, which ones are more desirable, and what might be an appropriate approach to completing the concept. The Use Study may also describe the expected or required experience and training level of support personnel; from this, the training needs of support personal may be established.

The Use Study may be a contractual document, as it sets the basic requirements for software support. The Use Study is important for the Supplier, who will ensure that the support concept complies with the Customer requirements. The Use Study should be flexible and should be reviewed for mutual agreement. From the basis of the Use Study, the Supplier can offer new technologies or more cost-efficient support options of which the customer may not be aware. The Use Study also permits the Customer to refine initial decisions and concepts that the analysis shows are impractical or extremely expensive. [DEFS0060-1] provides a detailed description of one possible Use Study format.

6.2 Software Identification and Breakdown—Any software supportability analysis process has to determine how all software within a system is, or is not, to be supported. Thus, it is essential that all items of software be uniquely identified, preferably in a coherent and systematic way that also 'fits' with the system and hardware identification used for design, production and logistics purposes. Uniquely identified software items are normally subject to configuration control and are thus known as Configuration Control Items (CCIs). Item identification numbers are often known as Logistic Control Numbers (LCNs).

Existing methods for software Logistic Support Analysis Control Numbers, such as defined in [DEFS0060-3] and [MILPRF49506], are evolving in their application to software, but do not yet provide as complete traceability as is desired. An extended method, providing traceability between software functions, their physical representation as programs or files and the hardware implementing the software functions is presented here. See reference [HUTCH00] for a detailed description of this method. This method is compatible with general logistic numbering conventions, including the [DEFS0060-3], given by the standard Logistic Support Analysis (LSA) process.

It is important to clarify guidance criteria to be used for this association of software and hardware within a system. The identification method is structured in order to permit different levels of abstraction. Depending on the component relationships that are going to be addressed, this structure will either be:

a. A 'functional' breakdown, i.e., a design breakdown of the different modules; or,
b. A 'physical' breakdown, i.e., the 'loadable' components provided on a transfer medium to the user.

Depending on the component relationships to be addressed, one of the two or both breakdowns may be required. A mapping between both breakdowns may also be required for logistics management support, in order to understand how hardware and software support activities are interrelated. For example, changes and/or repairs to a hardware component may require reloading associated software components or even modifying the functionality of the software. The physical breakdown link to software components would alert the support function to review those software components for impacts and further support actions.

Data that is configured and controlled should be identified similarly to application software and hardware. In this case, a data breakdown from the 'engineering' or logical point of view should be considered based on its individual functional components. For operational purposes it may be appropriate to break down data sets on the basis of how they are physically prepared, loaded and/or manipulated.

6.2.1 GUIDELINES FOR THE FUNCTIONAL BREAKDOWN—The 'functional' breakdown includes all those elements that depict the functional design aspects of interest to the system software engineers for Modification Support. See Figure 9 for an example.

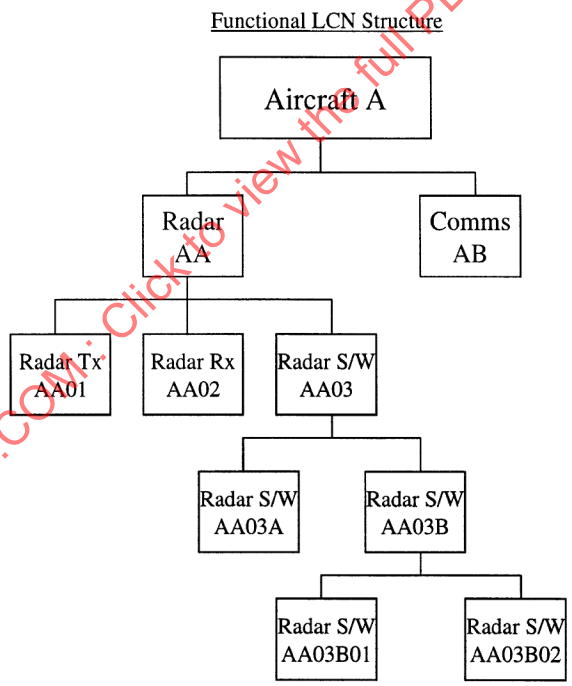


FIGURE 9—FUNCTIONAL BREAKDOWN EXAMPLE

The functional identification breakdown starts with the system and breaks this system down into its constituent functional components. The breakdown should be continued until the lowest level function requiring support is identified. Where a software component exists within a system, it should be identified as an individual component. Therefore, in Figure 9, the Radar software provides both Transmit (Tx) and Receive (Rx) functionality, but is considered at the same level as the transmitter and receiver hardware components. From this point, the Radar software functionality is divided into Transmit and Receive functions and so on. The lowest levels of functionality considered are the software controlling the Radar display and the software carrying out the Radar Data Processing. The guideline is to make the functional breakdown and identification follow as closely as possible the actual system design.

The level of functional breakdown identification for logistic support depends on an initial maintenance significance analysis. Initially, only the system/subsystem and low-level Configuration Item levels are required. Later, depending on the depth of the analysis, the breakdown may be continued to Configuration Controlled Component level.

6.2.2 GUIDELINES FOR THE PHYSICAL BREAKDOWN—The 'physical' software corresponds to all those software elements (including data sets) that can be manipulated separately (e.g., loaded, downloaded, modified, and/or analyzed), as a single entity, by the maintainer, user or operator of that system. See Figure 10 for an example.

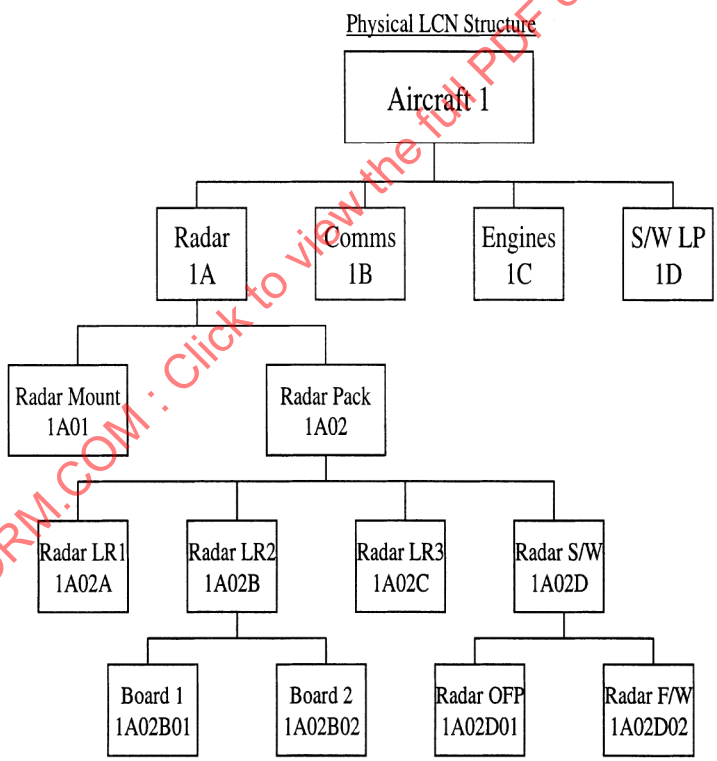


FIGURE 10—PHYSICAL BREAKDOWN EXAMPLE

These system elements are what the user actually 'sees', as opposed to the 'functional' elements representing what the system 'does'. As such, the software identified here is the software that has to be identified for the software support analysis and carefully coordinated with system logistic support analysis.

In Figure 10, the software is represented as part of the hardware system where it resides. The software is identified at the highest level at which it can be seen as a component of the system. In this case, the Radar is made up of a number of components, Line Replaceable Items (LRIs), and the Radar software. This software consists of 2 elements: an Operational Flight Program (OFP) loadable by the maintainer, user or operator and embedded, normally inaccessible, software (firmware). A further breakdown of the software could be carried out to identify individual files within an application. In parallel with this, the breakdown of the system hardware must be carried out to the level at which the associated software influences supportability.

6.2.3 GUIDELINES FOR PHYSICAL TO FUNCTIONAL MAPPING—The support concept for a system will be influenced as much by the method for supporting software as for the hardware. Therefore, it is important to have as complete a picture of the system as possible. While software functionality can be broken down to a low level, and the corresponding physical programs or files can be identified, it is only when software functions are considered together with the hardware on which the software resides that the picture is completed. Hardware-software interdependencies, such as processor type, memory requirements, software loading point and timing dependencies, need to be analyzed such that the effect of alternative technologies or maintenance activities are addressed. This is a 2-way process: hardware changes, such as adopting a new processor, may affect software components; changes to software components, whether corrective, adaptive or perfective, will have some effect on the system hardware, even if it is only for identification of the task of loading the new version. In order to fully identify the support resources and requirements of the system represented in Figures 9 and 10, the following information is required as illustrated by the relationships in Figure 11:

a. What physical software components implement the individual functions?
b. What physical hardware components does the software depend on?
c. Where in the system is the software loaded?

Any change to a physical or functional element requires consideration of the potential effect on the other related elements as defined by the arrow connections.

In the system in Figure 11, the support requirements of the 3 functional elements of radar software can be determined as follows:

a. The transmitter function software is implemented by firmware. The loading point for this code is directly into read-only memory, on a board within Line Replaceable Item (LRI) 2. In order to execute, the software module relies on some hardware aspect of another board within LRI 2.
b. The display software constitutes part of the OFP, loaded at the aircraft Software Loading Point (S/W LP). This software relies on the LRI 3 hardware.
c. The Radar Data Processing function is also part of the OFP, again, loaded at the aircraft Software Loading Point (S/W LP). This software relies on the LRI 1 hardware.

For more complex systems, multiple dependencies may exist, giving many-to-one or one-to-many relationships between physical and functional items. These relationships can be easily accommodated within the above system, allowing the supportability issues of the whole system to be addressed together.

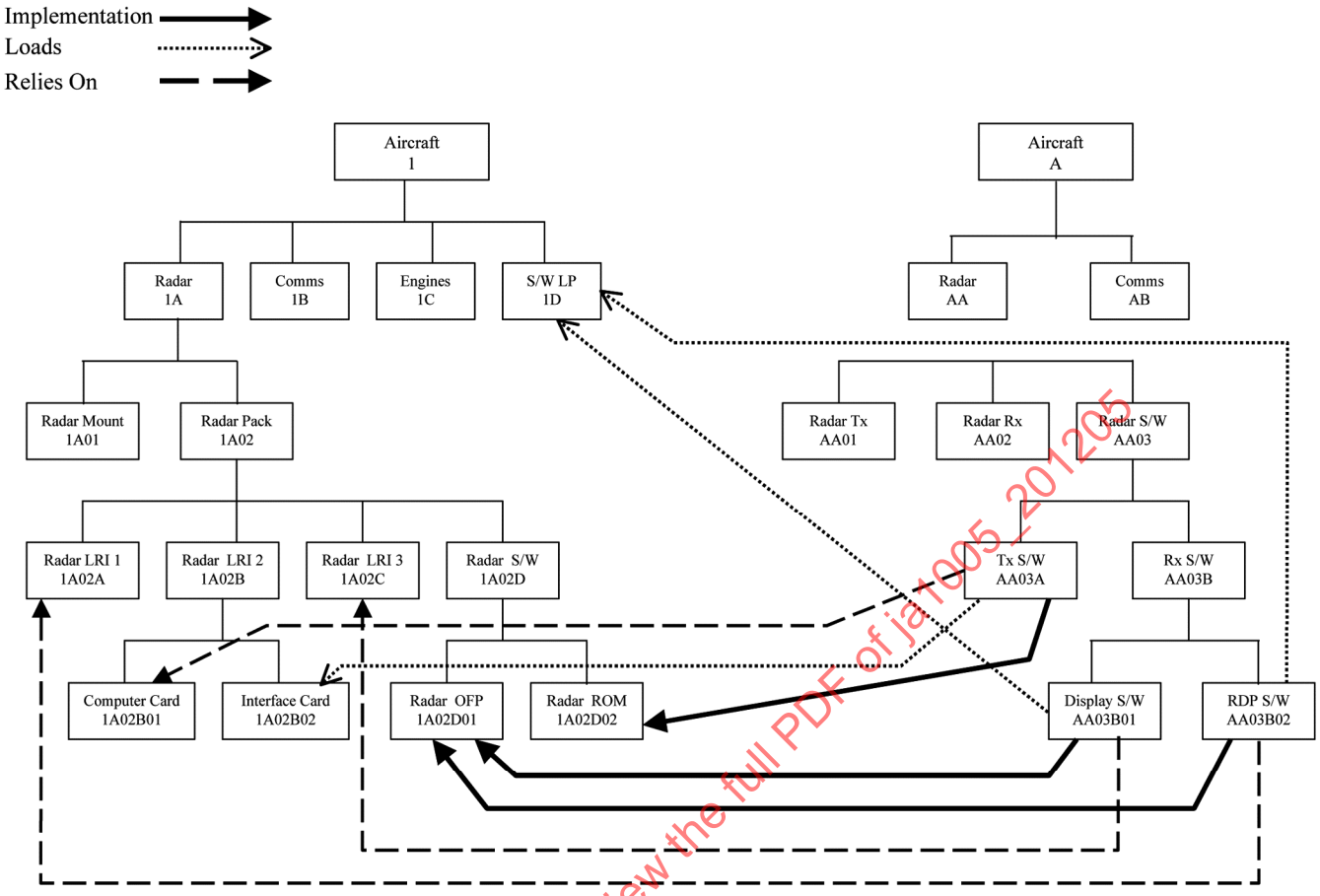


FIGURE 11—PHYSICAL TO FUNCTIONAL MAPPING

6.3 Categorization and Selection of Software Support Significant Items—Once the initial breakdown of software has been carried out, both for the functional and the physical aspects, each of the elements that have been identified have to be categorized and prioritized in terms of their support significance. An initial decision that has to be made for each software item is whether or not to proceed to a supportability program and analysis. Both developmental and non-developmental (e.g., OTS) software items should be categorized. In addition, the support significance of support system software (e.g., operating system, compilers, design tools, analysis tools) should be categorized.

A minimum set of information should be collected in order to cover those main parameters that could affect software support (e.g., operational importance, property rights, frequency of change). After evaluating all individual elements, software items would be categorized into one of three basic categories:

a. Not Applicable—Software does not merit further supportability analysis at this time;
b. Partial—Software should have some supportability analysis to collect a specified subset of information to augment a reasonably simple support concept; and
c. Potential—Software should be selected as a potential full support candidate and explore all possible support alternatives.

Potential software support candidates are subjected to tailored supportability analysis including, if a developmental item, their design-for-supportability characteristics. These candidates should be selected on the basis of their operational or supportability significance. It makes no sense, for example, to select as a potential supportability candidate a firmware product that is not expected to change within its host equipment's life cycle.

Three distinct types of support candidates should be considered:

a. Functional Software Support Candidates;
b. Physical Software Support Candidates; and
c. Data Candidates.

Each candidate type has its own characteristics and inherent supportability problems.

6.3.1 SELECTION OF FUNCTIONAL SOFTWARE SUPPORT CANDIDATES—Functional candidates should be all those software elements that either represent a system, a computer Configuration Controlled Item or those software items that have a design of their own (lower level Configuration Controlled Items), as all these might be modified separately and might hence have different support concepts associated to them. These elements are the ones considered for the Modification Support Analysis.

In addition to the above software items, it is convenient to assign as potential support candidates all software with product supportability risks, such as multiple languages, product versions/variations or multiple applications. Software that has non-standard design methods, is reused from existing applications, or is proprietary/security/safety sensitive should be also considered. The support environment might also influence the selection of potential candidates. A potential support significant candidate does not automatically mean that a full support concept will be developed for it, only that it merits closer inspection, including at least a partial supportability analysis.

6.3.2 SELECTION OF PHYSICAL SOFTWARE SUPPORT CANDIDATES—Physical candidates should be all those software executable modules that can be loaded and/or installed separately or that require different loading/ installation tasks. This does not mean that each individual library that is delivered with an executable program is a potential candidate. Common sense would dictate that groups of files (e.g., executables, libraries, help files, licenses) delivered or loaded/installed together in one single operation would be considered a single 'candidate', even if consisting of multiple components. The physical candidates are the ones considered for the Logistics Management Support and Operational Support Analysis.

6.3.3 SELECTION OF DATA SUPPORT CANDIDATES—Data supportability, due to its mainly electronic nature, is often assessed together with software. In this context, data items that are candidates for support do not differ radically from software candidates. There may be a logistic identification link between one or more data items and one or more software items that require supportability analysis as a group.

In principle, data support candidates should include all databases, as well as all data blocks that are either manipulated as a single entity (e.g., a sound track, a parameter file), that require external software for creation, manipulation or evaluation, or are installed, loaded or unloaded by means of a specific task (e.g., download mission data, mount data tape).

Functional data candidates are also possible, particularly when assessing complex data-intensive systems such as banking or production process/stock control applications. In case multiple programs access different subsets of a same database, or when multiple databases are cross-linked, the selection of functional data candidates usually will be determined by the need to ensure that changes to a data set because of the needs of a specific application (or vice versa) does not propagate unnecessarily to other systems or, even worse, makes those other systems inoperable.

6.4 Support Scenario Modeling—This section provides some guidance on a modeling method that may assist the implementation of support scenarios. At the most detailed level of the scenario, the task interrelationships, people, organizations, resources and infrastructure, and low-level procedures required to carry out the tasks are described.

The generation of support scenarios is a basic analytical tool for defining software support requirements and building a complete software support concept for a system. Performance parameters in terms of effort, response time, and frequency of tasks can be determined through scenario derivation and analysis. As support scenarios provide a means of linking support levels, support agents and functions, they establish the relationships among all the dimensions of the support concept framework (e.g., see Figure 3). Furthermore, since performance measures can be applied to elements within a support scenario, scenarios can facilitate testing of proposed support concepts against quality of service requirements. An excellent description of the varied uses and applications of scenarios for modeling operational use of systems is described in [WEID98].

6.4.1 SUPPORT SCENARIOS AND TRIGGER EVENTS—A support scenario is an integrated representation of resources and processes, showing how a desired end result will be achieved in response to an event that triggers a demand for a support function.

Defining trigger events is the first step in creating support scenarios. The following list indicates some typical types of trigger events that might be relevant for any particular system:

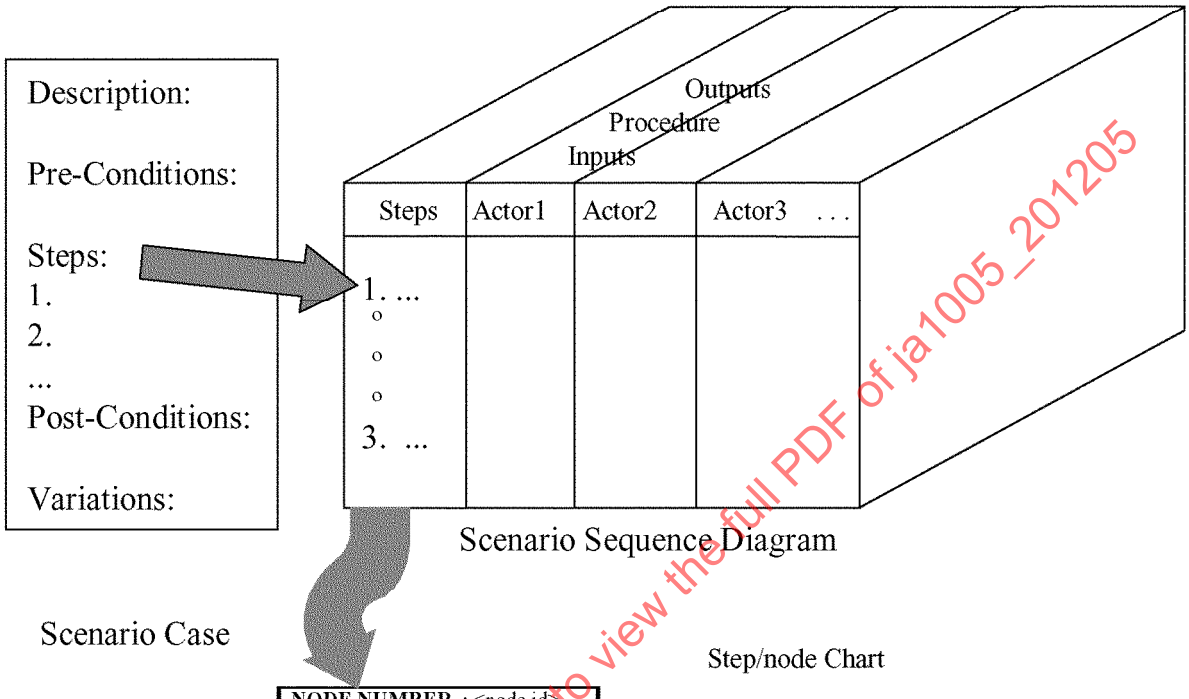
a. System/software failure;
b. Technology upgrade for support system, operational platform hardware, or data interfaces;
c. User help request on operational use of system;
d. Request for software enhancement (functional or performance);
e. Major disruption to Software Support Center Facilities (e.g., fire or extended loss of power supplies);
f. Mission preparation;
g. Mission completion; and
h. Scheduled system backup.

Based on the primary trigger events, support scenarios are generated to show the sequence of typical actions, information flows, resources used and procedures invoked. Trigger events arising in the 'system' environment should be the first priority, since these will cover the immediate requirements for maintaining the system in satisfactory operational condition. Scenarios should also be considered for trigger events that arise in the 'support' environment. The analysis of trigger events should also be used to expose the through-life technical and financial risks that must be mitigated in logistic management plans and procedures.

The process of generating and analyzing software support scenarios lends itself to modeling with computer-based simulation or operational analysis tools. A customer and supplier, as part of the overall approach to supportability analysis for a project, should jointly consider the possible use of such tools.

Support scenarios are intrinsically linked to the evolving support concept description for a system. Hence, it is necessary for a supportability program to establish a link between support scenario descriptions and the discrete tasks that will in due course actually deliver the required support functions and services. For any particular system there will be many potential support scenarios, and it would be possible to generate a discrete model for every trigger event. However, in practice this is unlikely to be necessary or efficient. Support scenarios which have a high degree of commonality in their resource and process elements should be combined, with the inclusion of appropriate decision points, alternative paths and other suitable conventions.

6.4.2 REPRESENTATION OF SUPPORT SCENARIOS—It goes beyond the scope of this document to discuss the variety of different techniques used for business and/or process modeling, but the general methods of the Rational Unified Process [ROYCE98] for the Use Case, Sequence Diagram, and Class Diagram have been adapted to the support scenario application artifacts: Scenario Case, Scenario Sequence Diagram, and Scenario Step/Node Chart as illustrated in Figure 12. Depending on the required understanding, these artifacts provide increasing level of detail about the scenario. A few examples of partial scenario applications are illustrated in Appendix C.



NODE NUMBER : <node id>			
AGENT TYPE : <e.g., user, operator, supporter>		LEVEL : <e.g., field, intermediate, depot, vendor>	
SUPPORT FUNCTION NAME : <short name for support function>		SUPPORT FUNCTION TYPE: <e.g., operational, logistics management, modification>	
INPUTS		OUTPUTS	
From Node	Description	To Node	Description
<previous node id>	<list/description of input data, forms, actions>	<next node id>	<list/description of output data, forms, actions>
TASKS A. < List of actions, activities, procedures conducted as part of this node's function.>			

FIGURE 12—SUPPORT SCENARIO MODEL FRAMEWORK

6.4.2.1 *Scenario Case*—At the initial stage of generation, a support scenario might well not have all the relevant data precisely defined - on Support Agents and Levels, for instance. However, it will be possible to refine this data as more scenarios are considered and patterns emerge of the optimum groupings for major functions.

Hence, given a specific trigger event, the first scenario artifact that might be produced is called the 'scenario case'. Preconditions for the trigger event to occur are described. Each of the successive primary flow steps expected to occur are described. The post-conditions expected after all steps have been completed are also described. For any steps where the primary action might have variations, those variations are briefly described.

This initial scenario view can usually be produced fairly quickly and provides a good basis for discussing software support activities at a high level without getting over-involved in too much detail.

6.4.2.2 *Scenario Sequence Diagram*—Once a scenario case has been generated, it is now possible to include more detail for each of the sequential support steps. In particular, it is of interest to determine at each sequence task step:

a. What lower-level activities are performed?
b. Where are these activities performed?
c. Who performs the activities and what skills are required.
d. What tools and support equipment are necessary to perform the activities?
e. What performance (effort, response time) is expected and/or required.

The scenario sequence diagram provides a modeling artifact to capture this information by linking each step to the appropriate information in the sequence diagram. The sequence diagram, as illustrated in Figure 12, provides the modeling capability to identify support agent actors/levels and the lower-level sequences of activities that occur at each step. Variances within the lower-level sequences are also possible.

6.4.2.3 *Scenario Step/Node Diagram*—If lower-level inputs, outputs, and detailed procedures are appropriate for the individual activity nodes in the scenario sequence diagram, then node charts such as illustrated in Figure 12 are derived. The specific procedures for each node can indicate what equipment and specialized knowledge may be required, information that may be very specific to individual software items. At this level of information, essentially all elements of the software support concept dimensions for this specific scenario are now known.

6.5 Block Release Profile Modeling—Whenever there is a need to change any of the baseline system software items, the updated software should be released in accordance with a Block Release Process. An example of a general Software Support Process Flow is illustrated in Figure 13.

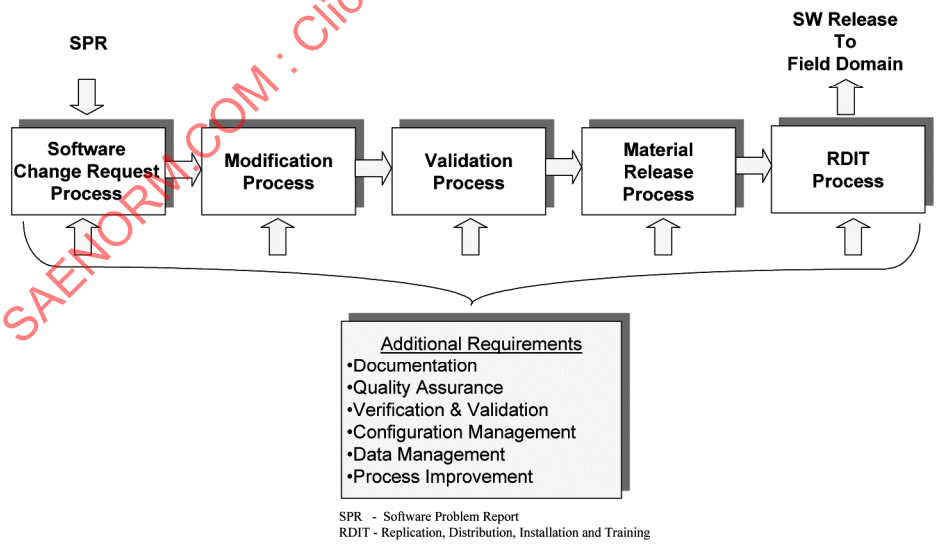


FIGURE 13—SOFTWARE SUPPORT PROCESS FLOW

Software changes will result from a need to correct software defects, enhance software functional and performance capabilities, and adapt software to hardware and data environment changes. A software change is documented and submitted as a Software Change Request (SCR), and approved by the Configuration Control Board. See Figure 14 for an example SCR Process Flow. In this figure, "Customer Engineering" represents the support engineering interface between Customers and the Supplier contractors, vendors, and perhaps in-house engineering support staff. One or more of the approved SCRs are then scheduled for implementation into a future Block Release of the system software. A Block Release may implement one or more SCRs.

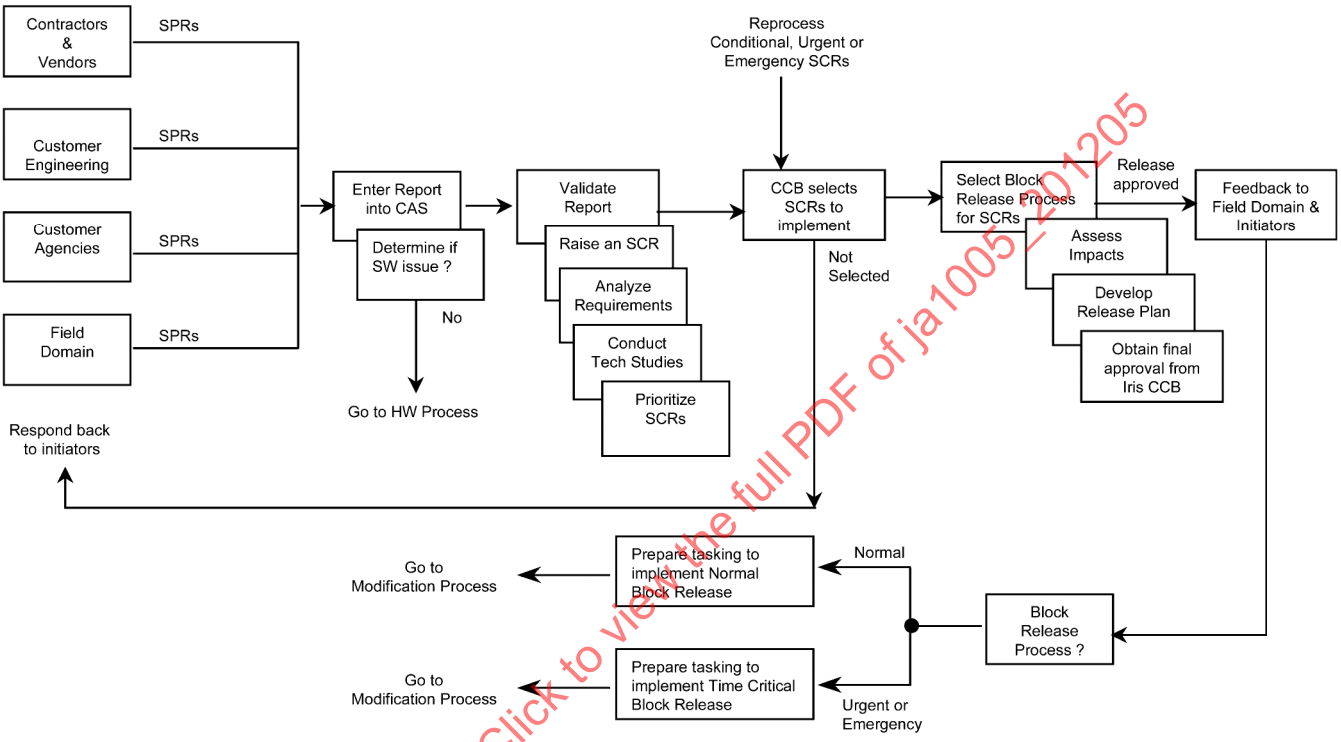


FIGURE 14—CHANGE REQUEST PROCESS FLOW

There are at least three distinct types of Block Releases for the System:

a. Normal;
b. Interim; and
c. Emergency.

6.5.1 NORMAL BLOCK RELEASE—Typically, a Normal Block Release will implement several SCRs that solve many problems, and include all the steps in the Process Flow as illustrated in Figure 13. SCRs that are not time critical, as well as completion of time-critical SCRs that were completed through urgent or emergency block releases, are candidates to be fully implemented under the Normal Block Release Process. The schedule required to implement a Normal Block Release depends upon the time required to execute all the steps in the Process Flow and the time required by the Field Domain to effectively adapt to the changes provided by the previous Normal Block Release. An example of timing for the development of a normal block release might be as shown in Table 1:

TABLE 1—EXAMPLE NORMAL BLOCK RELEASE SCHEDULE

Process Step	Timing
Software Change Request (package of requests)	3 months
Modification	9 months
Validation	3 months
Material Release	1 month
Replication, Distribution, Installation, and Training (RDIT)	1 month
Totals	17 months

Thus, if it were desired to release consecutive normal block releases, one would be released every 17 months. On the other hand, if it were desirable to field a block release every 12 months, then the releases would have to be overlapped on an iterative build cycle. An example of block release data of interest (see [PEERCY2] and [DEFS0060-3] for more complete information) includes the following items for each normal block release:

a. Type Release – Normal, Interim, Emergency
b. Total # SCRs – Total Number of Change Requests implemented by the block release
c. # SCRs by Type – Number of Change Requests by Type (Corrective, Perfective, Adaptive)
d. # SCRs By Complexity – Number of Change Requests by Complexity (High, Medium, Low)
e. # SCRs by Priority – Number of Change Requests by Priority (Normal, Urgent, Emergency)
f. Personnel Load by Function – Personnel time (e.g., person hours, person months, equivalent full time equivalent) on the release by functional area (e.g., change analysis, modification, validation, material release, RDIT)
g. Personnel Skill Level – Measure of overall personnel capability to complete release by functional area (e.g., change analysis, modification, validation, material release, RDIT)
h. Changed Source Size – Measure of the amount of software changed by the implemented Change Requests (e.g., Function Points, Source Lines of Code)
i. Schedule – Breakdown of block releases schedule timing by functional area (e.g., change analysis, modification, validation, material release, RDIT); includes specification of overlap time with other releases

6.5.2 INTERIM BLOCK RELEASE—The purpose of the Interim Block Release Process is to provide an acceptable operational responsiveness to a requested software change when the following occurs:

a. One or more Process Flow steps have not been successfully passed; and/or;
b. An immediate Block Release is required to overcome difficulties experienced by the Field Domain.

For the case where a Block Release has not successfully passed one or more Process Flow steps, the Interim Block Release process enables Customer Engineering to salvage the Normal Block Release and obtain an interim capability that overcomes some of the difficulties experienced by the Field Domain.

For the case where an immediate Block Release is required to overcome difficulties experienced by the Field Domain and a solution has already been implemented that is available, Customer Engineering could issue an Interim Block Release that contains the implemented SCR solutions.

All software changes provided by an Interim Block Release are part of the next available Normal Block Release in order to ensure a permanent solution is implemented within the baseline of the system.

6.5.3 EMERGENCY RELEASE—The purpose of the Emergency Block Release Process is to overcome a difficulty with the System within an allocated response time constraint. There are two types of Emergency Block Releases:

a. Urgent, this type of Release solves a specific problem within a maximum response time interval of an agreed-upon performance time (less than a Normal Block Release time but more than an Emergency Block Release time, e.g., 4 months).
b. Emergency, this type of Release implements a single SCR within a maximum response time interval of an agreed-upon performance time (e.g., 72 hours).

All Urgent and Emergency Releases are processed as illustrated in Figure 15, and included as part of the next available Normal Block Release in order to ensure a permanent solution is implemented within the baseline of the system.

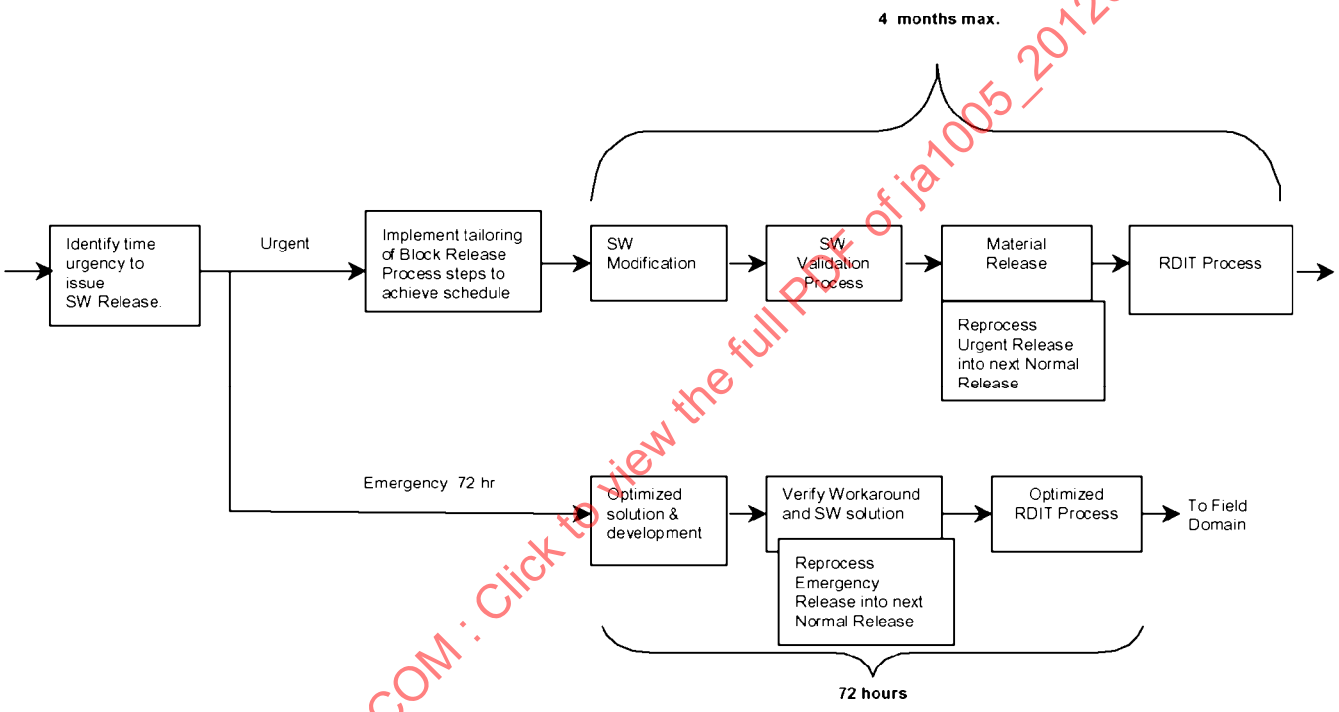


FIGURE 15—EMERGENCY BLOCK RELEASE PROCESS

6.5.4 BLOCK RELEASE IMPLEMENTATION METHOD—In order to be responsive and flexible to Block Release change schedule perturbations, it is useful to have an implementation concept that can accommodate short turnaround engineering cycles, inclusion of urgent change requests in the next Normal Block Release even though the Block Release is in progress, and emergency situations where a quick fix to a software item in a limited number of Field Domain equipment may be required. An extended Block Release schedule in the above-mentioned instances becomes unacceptable.

One model that can implement such a concept is the iterative build model illustrated in Figure 16 where individual changes are being implemented in smaller overlapping build cycles so that a Block Release can be accomplished rapidly. The advantages of this approach are maximum effective use of scarce and costly software engineering personnel resources, and a reduction in the Block Release schedule as compared to producing sequential Block Releases. This approach represents the maximum degree of overlap possible when using a single software engineering support agency.

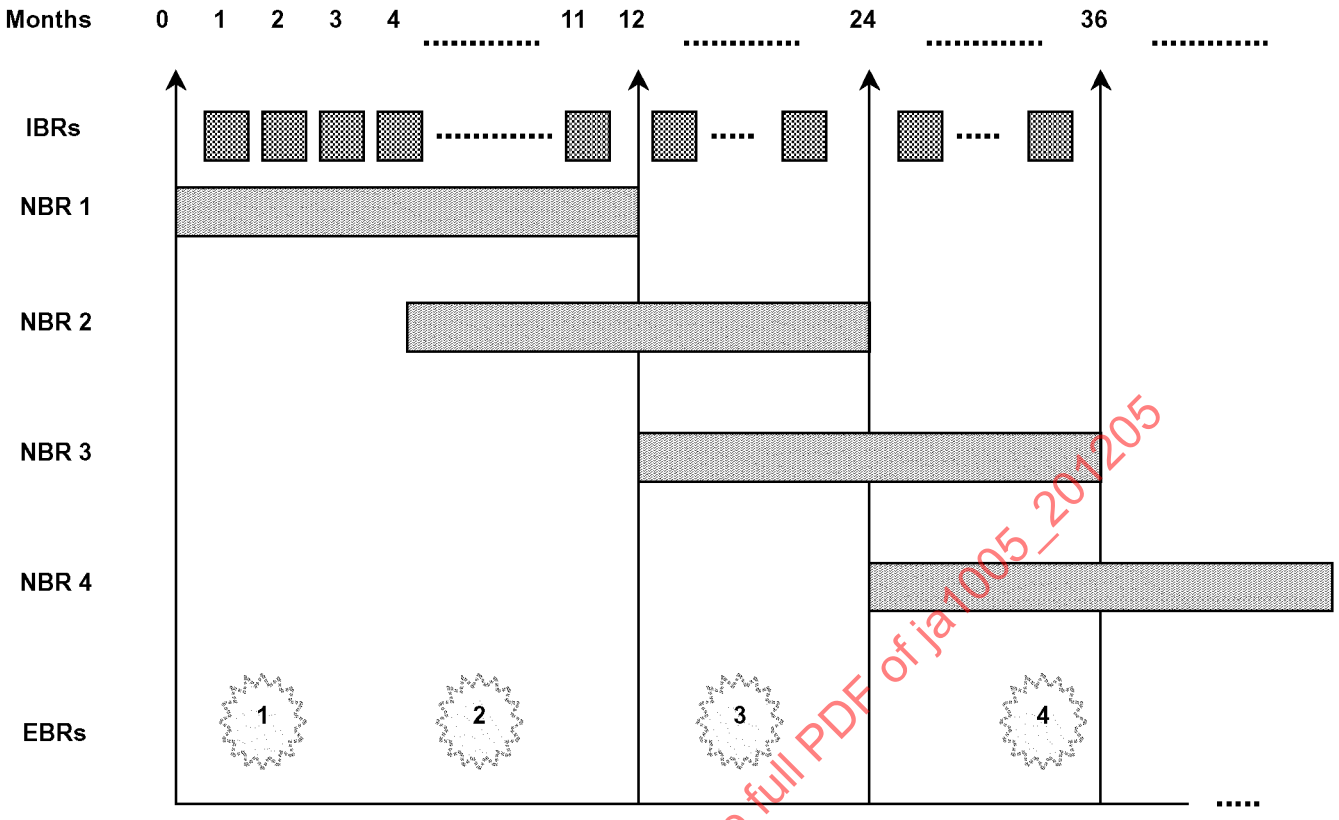


FIGURE 16—ITERATIVE BLOCK RELEASE EXAMPLE SCHEDULE

The Block Release process follows a schedule-bound process, where new Normal Block Releases (NBRs) are released on a periodic basis, and contain as many implemented (and prioritized) SCRs as resources permit. As an example, in order to achieve periodic NBRs on a desired 12-month cycle, an approximate work period of 20 to 24 months is estimated to be required for each NBR. The overlap of each 20 to 24 month NBR results in a 12-month apparent NBR cycle.

Every specified period of time (e.g., 30 days), Customer Engineering has access to all implemented SCRs to-date, which can be made available through the issuance of an Interim Block Release (IBR). This enables Customer Engineering to immediately provide an already implemented solution should difficulties be experienced by the Field Domain.

From time to time, Customer Engineering will produce Emergency Block Releases (EBRs) that are required to provide solutions when time responsiveness is critical. As an example, it might be estimated that a maximum of two EBRs per 12 calendar months would be required during the life of the System.

The Block Release Implementation Method would need to be tailored to specific Customer/Supplier organizations based on the Software Support Concept derived to meet Customer performance requirements.

6.6 Life Cycle Costs and Trade-Offs—Any software supportability study must of course address the life cycle cost of the software to be developed, fielded and supported. Though a lot of consideration is frequently given to the software development costs, all too often the cost of support is forgotten, despite, by accepted wisdom, this being by far the most expensive part of the software life cycle. The current consensus is that software maintenance accounts for as much as – or perhaps even more than – 80% of the software life cycle. However, this figure does not consider the additional cost associated to software, such as the training and occupation of the operators of such software, nor additional costs such as software distribution or the production of software manuals; nor the direct economic cost of software faults on software operation, and the time it might take to resolve those faults. When this is taken into account, the organizations developing software or acquiring software should very seriously consider whether the spent effort and money has not been just the prelude to something even more expensive.

Under these circumstances, carrying out a software supportability program, including a supportability analysis during the early study phases and in parallel to the requirements definition and software design, might ensure that supportability is appropriately considered, that proper trade-offs are carried out against different operational and support solutions, and that most cost-effective alternatives are ultimately selected. Such a supportability program has to ensure that the necessary software supportability is built into the system and, while costing only a fraction of the development cost, an equivalent reduction in support costs should greatly reduce the cost of ownership. A well-managed software supportability program - especially in high-cost systems - usually pays for itself.

6.6.1 SOFTWARE COST ESTIMATION MODELS—A primary factor in project management is that of time and cost estimation. Software cost estimation models can be used to provide the required time and cost estimates together with an assessment of the significant cost drivers. Many software cost estimation models are available, either in the public domain or as part of commercial tools (e.g., COCOMO, PRICE-S, SEER-SEM, SLIM, REVIC, SASET). An outline [SEP02], description and link to both public domain and commercial tool web sites can be found at [SEPO] (Software Planning and Estimation selection). The majority of software cost estimation models are parametric models in which time and effort estimates are calculated as a function of a number of parameters. The number of parameters used varies widely between models. Models where the algorithms used are given are typically known as 'open' models. Many models are 'closed', that is, the time and cost algorithms are not given. Many models operate in 2 stages: the first stage estimates the size of the software project usually in terms of source lines of code or function points. The second stage estimates the time and effort required to produce the estimated size of software.

However, software cost estimation models in most cases address only the required effort and schedule, and very often only for software development. The cost is usually computed on the basis of necessary person-months. Additional costs such as training needs, documentation, computing infrastructure or integration rigs can have a significant cost impact yet are rarely considered in the existing cost models. Similarly, the logistics aspects (e.g., replication, packaging, distribution) or the operational aspects (e.g., installation, configuration) can have also a significant cost, yet are again rarely considered by the model. These costs are often recurrent, and are to be considered over the whole life of a specific piece of software.

As illustrated in Figure 17, the life cycle cost of software consists of the sum of:

a. Cost of Software Development;
b. Cost for Software Modification Support;
c. Cost for Software Logistics Management Support;
d. Cost for Software Operational Support;
e. Infrastructure Costs, including the maintenance of such infrastructure; and
f. Other Costs (e.g., travel expenses, indirect costs, financial expenses).

Software Support Costs include all except those related to the Software Development.

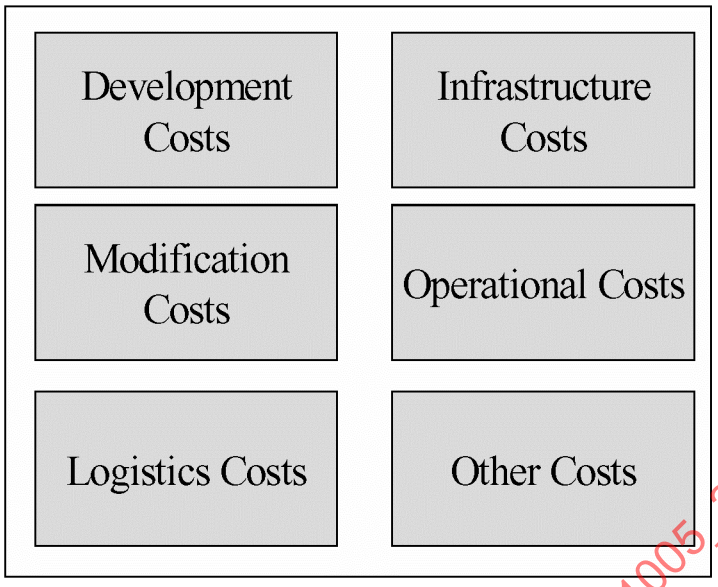


FIGURE 17—SOFTWARE LIFE CYCLE COST CATEGORIES

6.6.1.1 Development and Modification Costs—Development and Modification costs can, with some degree of confidence, be estimated by means of an existing cost model, such as those outlined above. The most popular cost models are based on the COCOMO 81 model described by Barry Boehm in his classic work [BOEHM81]. This model, however, is not well suited for the assessment of modification costs. An evolved model, COCOMO II, does consider the peculiarities of modification costs. Further information on this model as well as actual implementations on different platforms can be found at [USC]. PRICE-S is another particularly popular cost estimation model.

Caution should be however exercised with the use of any of these tools. For example, many models assume a certain development method, usually the typical waterfall model, which might not necessarily be the program development method. Similarly, not all models and/or tools cover all development phases, such as requirements analysis and/or system integration. Before using the results of such tools, it is essential to ascertain what aspects and phases of the software life cycle are covered and what assumptions the model makes. Moreover, several models require calibration using historical data on past projects of the organization. It is not valid to calibrate models using data from other organizations; moreover, data required for calibration is often unavailable. Uncalibrated models produce less accurate results.

Independently of the actual model used, a value will be obtained describing the effort required for either the development, or maintenance, or both. This effort – typically given in person-months – can be converted into an economic cost. Most existing cost models also provide a development schedule.

It should also be noted that most models only consider staff effort, and neglect the cost of the necessary infrastructure (office space, special facilities). The required hardware (host computers, hot benches) or necessary software licenses (compilers, analysis tools) are usually not considered. While it is possible to consider these costs here, in most cases these are considered under the infrastructure cost element in 6.6.1.4.

Some experimental models are available for special needs. For example, at [USC] there is also a COCOMO-derived model oriented towards the integration of Commercial Off –The-Shelf (COTS) software. Caution must be exercised when using any cost model, particularly newer or experimental models, to ensure that the models underlying assumptions and formulae have been fully validated and provide software support related estimation capabilities.

6.6.1.2 *Logistics Costs*—There are no specific Software Logistics Management Support Cost Models currently available, but some of these costs can be determined in a similar way as for hardware logistics. In this particular case, the 'spares' are only replacements for defective or damaged media. Though typical cost estimation tools used for hardware production can be used for this purpose, it should however be kept in mind that the replication, packaging and distribution is repeated over and over with every new software version. Not only this, but each new software version may require its own documentation and possibly even some delta training for the affected users. It might well be that hardware is manufactured and distributed only once in a product's life, but many software versions might eventually run on that hardware. One of the critical cost parameters for the logistics cost is therefore the number of software versions to be produced and distributed over the system's life cycle.

Another significant support cost may be the 'help desk' that is required to provide the users with a focal point for software-related problems. Depending on the project and application type, this help desk may or not be shared with other system or even pure hardware-related aspects. The cost of this help desk strongly depends on the quality of the software product, as well as on the number of users of that product. A defective or badly documented product will tend to result in more problem reports and/or user queries. Similarly, a user base of hundreds of thousands or even millions of potential users, such as for many commercial packages, will generate a greater feedback than a very specialized product with a limited user base.

6.6.1.3 *Operational Costs*—There are currently no Operational Cost models oriented specifically for software support. However, experience shows that typical tools used for hardware, and in particular for elements such as Level of Repair Analysis (LORA) may be suited for this purpose. It should be also kept in mind that, quite often, operational aspects such as software loading and/or installation after repairing a processor module have to be combined with hardware-related tasks. It is important to detect all support triggers, operational tasks and associated task frequencies to ensure that all costs are properly considered. One useful method for this purpose is a Failure Modes, Effects and Criticality Analysis (FMECA) [ARP5580], [IEC60812], [BS5760] to determine which types of failures may lead to such operational tasks. However, a FMECA does not cover all possibilities, such as for example new software versions or re-configuration of the software for purposes other than a failure. In this context, Support Scenario modeling provides a fuller perspective, provided the inputs from the FMECA are properly considered.

Operational costs should cover both the cost, if applicable, of installing new software upgrades in all appropriate in-service equipment together with the cost, if applicable, of reloading software due to the repair of the equipment where it resides.

The operational costs can often be reduced by combining hardware and software related tasks, or ensuring that different support triggers (e.g., hardware repair, new software baseline) are channeled towards a same or compatible solution (e.g., same loading level) that permits sharing of necessary resources (e.g., loading equipment, repair technicians, operator training courses). If such policy is established, the additional operational cost of software may not be significant.

6.6.1.4 *Infrastructure Costs*—Up to now, only the cost of carrying out the individual support functions has been considered. Such costs may be only part of the story. Infrastructure costs for support systems, testing environments, and special facilities may amount to a significant part of the software support costs, depending on the application.

The cost of certain resources such as host computers and/or replication facilities may not be costed under the appropriate functional support area. The reason for this is that such resources require special facilities (e.g., air conditioning rooms, office space) that form part of the support infrastructure. Thus, often these (usually hardware) resources are sometimes considered infrastructure themselves, and are therefore accounted as assets, in the same way as a building would be.

The cost of such resources may be high, especially in the case of safety-critical software that has to comply with a standard such as [RTCADO178B] or equivalent. The simulation and testing environment (say, an avionics integration bench) for the safety demonstration can easily cost tens of millions of dollars. On the other hand, it should be remembered that while the acquisition is a non-recurrent cost, the standard maintenance costs for such an environment are recurrent and such maintenance has to be paid annually for the whole life cycle of this support environment in order to keep it operating. The total life cycle bill might be prohibitively high.

Similarly, the environment where such resources are housed often has a significant cost. Office space is expensive, particularly if special environments for, and requirements on, software testing imply special facilities such as explosion-proof rooms, for example if testing high-pressure systems controlled by software.

It is however not always possible nor appropriate to put these infrastructure costs on the 'software' side of the support cost equation. While such infrastructure is often required to test new software, it is also used to evaluate system and/or hardware/software integration issues and/or test proposed modifications that are 'hardware only'. The decision of where the actual cost 'burden' is placed is often based on the existing corporate accounting practices or contractual arrangements.

Infrastructure costs also account for all costs that have to be paid in order to have a location, a facility, support equipment, software tools and staff ready to carry out support, independently of whether this support is effectively carried out or not. Thus, a telephone line is an infrastructure cost for a help desk, independently of whether users call in or not. The mere fact of having the help desk available requires some investment – training for support staff, for example – and recurrent costs – telephone bills, office rent, electricity, maintenance of equipment, and so forth. No users may actually call in, and the support staff may actually be doing other things in the meantime – but the costs remain.

All the above is true for all functional support areas. The mere fact of being capable of carrying out support functions costs money, independent of whether support is carried out or not. Thus, the support concept should take into special consideration whether such capabilities are actually required.

6.6.1.5 *Other Costs*—After assessment of infrastructure support costs other costs may still remain. There may be some fairly significant “hidden costs” associated with inadequate documentation, incomplete communication of software specifications to the customer, high-level management intervention when problems arise, and requirements for in-service presentations and additional meetings beyond those planned. Usually it is difficult to pinpoint these additional costs, and they cannot always be traced back to, or even directly associated with, software within a project, unless only software is involved. Other costs that should be considered include:

a. Travel costs;
b. Indirect costs and expenses due to unplanned risk items; and
c. Financial and legal costs.

6.6.2 METHODOLOGY FOR THE DETERMINATION OF SOFTWARE SUPPORT COSTS—The determination of software support costs can be carried out by a sequence of steps in which each of the different costs is determined. This sequence is interleaved with the rest of the support analysis, where the results of the analysis are inputs for this cost assessment and the outputs of this cost assessment are fed back to the analysis in order to carry out trade-offs that will permit the reduction of the support costs. The sequence for this cost analysis is as follows:

a. Step 1: Determine Cost-Significant Support Scenario;
b. Step 2: Determine Frequency for each individual task in each Scenario;
c. Step 3: Identify effort (staff) and required resources for each task in each Scenario;
d. Step 4: Determine stratified support costs for each task, each Scenario and each Support Function; and
e. Step 5: Feedback to Support Analysis for trade-off and optimization.

6.6.3 TRADE-OFFS—Supportability analysis should include an evaluation of both alternative software designs and alternative support concepts that result from an analysis of a 'trade-off' between alternative solutions. The overall aim of supportability trade-off analysis is to obtain an optimal balance between supportability costs and supportability effectiveness. Trade-off analysis at the system and/or software design level must take account of the impact of any design changes on software support. Factors for trade-off analysis include:

a. Response Times including diagnostic and transportation factors;
b. Cost of Ownership;
c. Effort;
d. Staff levels, staff experience and staff training;
e. Operational Impact;
f. Required Investment;
g. Technological interests; and
h. Limitations and known constraints.

Trade-off analysis can be used to assess the necessary staff effort and duration of both the development and the different software maintenance release blocks. If the analysis can also determine the relative length and effort of the different modification tasks, it will also determine the quantity and usage rate of other resources (e.g., support equipment, software tools) whose use has been identified by the supportability analysis. This usage can be quantified in terms of costs due to necessary investments, maintenance, or shared use with other programs. By studying the overlap between different release blocks and assessing the overall staffing ratio and equipment usage, it is possible to spread these release blocks in the most cost-efficient manner, by avoiding an excessive load on the required infrastructure while maintaining a constant staffing level. As supportability trade-offs are undertaken, it is very important that the supportability factors being traded are measured against each other in order to maintain a proper balance between various system requirements such as performance and availability capabilities, and modularity and commonality requirements. [DEFS0060-3] describes trade-off analysis in detail.

6.7 Design for Supportability—One of the dimensions of the support concept is the support classes. This dimension provides the designed-in characteristics of the software product, support processes, and support environment that enhance supportability. Designing for supportability includes:

a. Identifying which characteristics provide for better supportability;
b. Selecting those characteristics that can be designed-into the software support system and provide the highest impact; and
c. Ensuring those characteristics are appropriately designed-in and sustained or improved throughout the software life cycle.

At any point in the software life cycle, such design for supportability considerations can be applied, but the more effective approach is done earlier. Supportability analysis tasks can be conducted to identify support drivers and assess their implementation.

Maintainability characteristics of software products such as modularity, simplicity, descriptiveness, consistency, testability, instrumentation, are described in various references such as [AFOTECP3], [PEERCY2], [LMEPI140-xx], and [DEFS0060-3]. High level languages, object-oriented design and coding practices, and inspection review processes that focus on maintainability characteristics can have a large impact on the supportability of software products.

Characteristics of support processes that aid supportability can be found in references such as [AFOTECP2], [PEERCY1], [LMEPI140-xx], [SEICMM], [ISO15504], [IEEE1219], [ISO12207], [IEEE12207-0], [CECOM95], [MILHDBK347], and [DEFS0060-3].

Characteristics of support environment that aid supportability can be found in references such as [AFOTECP5], [PEERCY1], [LMEPI140-xx], [MILHDBK347], [MILHDBK1467], [CECOM95], and [DEFS0060-3].

Measures of maintenance capability and maintainability characteristics can also be found throughout the literature. For example, modularity and simplicity of object-oriented software design and code can be measured through the use of metrics such as those described in reference [MOOD98]. Metrics for the engineering principles of encapsulation, inheritance, coupling, and dynamic binding (polymorphism) are evaluated and related to modularity and simplicity.

6.8 Supportability Assessment—There are two primary forms of supportability assessment: assessment validation of the system support concept using results from test and evaluation; and assessment of operational, maintenance and logistic data after the system has been in service in its normal operating environment for a period of time:

a. Pre-service Supportability Assessment: Data from test and evaluation should be used to assist with the verification of system design and resources allocation decisions given in the Support Concept before the system is released to service. A successful pre-service supportability assessment may form part of contractual acceptance tests.
b. In-service Supportability Assessment: Data on system and/or report failures, software change request profiles, block release profiles should be assessed to verify that supportability requirements have been met.

The references [AFOTECP2], [AFOTECP3], [AFOTECP5], [PEERCY1], and [PEERCY2] provide detailed information concerning approaches to assessing the supportability characteristics of software life cycle processes, product, support environment, and supportability risk.

6.9 Interdiscipline Methods and Techniques

6.9.1 RELIABILITY-BASED ESTIMATION OF CORRECTIVE MAINTENANCE—By using software reliability estimation and prediction methods an estimate of remaining defects and the rate at which failures will occur (or are occurring) during the support activity can be made. Special attention may be warranted for potential safety-critical failures. This information can be integrated with appropriate configuration board control decisions to estimate how much corrective maintenance workload is likely during support. For example, suppose the reliability model predicts a total number of defects should be found during integration, system acceptance, and system operation over the life of the specified software package. With the number of defects actually found, delivered failure intensity rate of software for each operational hour, and information about expected operational time for the software, then a prediction can be made of how many defects will be found during each increment of time during operational use. Those expected defects plus planned enhancements and adaptations can then be distributed across the planned block releases for the software. Profiles of the change requests across priority, complexity, and type (corrective, perfective, adaptive) can more easily be planned based on the labor resources available and predicted corrective changes to be made.

The papers [SCHN98] and [SCHN99] provide a good discussion of applying reliability methods for estimating maintenance activity. Other references such as [LYU96], [MUSA99] and [MUSA92] also provide discussions on application of reliability principles for maintenance analysis purposes. [SAE JA1002] and [DEFS00-42] outline how to ensure software reliability throughout the maintenance activity.

6.9.2 FAILURE MODES, EFFECTS, AND CRITICALITY ANALYSIS (FMECA)—The Failure Modes Effects and Criticality Analysis (FMECA) analytically aims to identify all the probable modes of software failure, the cause of the failure, the possible effect of each failure, and the criticality of each effect on the system. The FMECA is a design analysis procedure and should be performed early in the design phase to aid in the evaluation of the design and to provide a basis for establishing corrective action. The FMECA procedure consists of 2 steps: a Failure Modes and Effects Analysis (FMEA) followed by a Criticality Analysis (CA).

FMECA is pertinent to software since an analysis at the system level should, for each failure mode, identify whether the cause of the failure is due, in any part, to the system software.

FMEA and \FMECA are described in [ARP5580], [IEC60812], and [DEFS0041].

6.9.3 FAILURE REPORTING, ANALYSIS, AND CORRECTIVE ACTION SYSTEM (FRACAS)—The FRACAS is an analytical procedure for evaluating system performance and the effectiveness of the logistics support process. Historical data captured with the FRACAS is relevant to the design and development of new systems. The FRACAS provides a mechanism for the collection of system failure data, identification of root cause of the failure and the cost implication of the failure, and the determination of the implied cost of failures. The FRACAS process is described in [DEFS0041] and [MILHDBK2155].

An extension to FRACAS is DRACAS (Data Reporting, Analysis, and Corrective Action System). DRACAS applies the analytical procedures of FRACAS to a greater scope of data, not just failure data, associated with the system. DRACAS is described in [DEFS0040-1].

6.9.4 LEVEL OF REPAIR ANALYSIS (LORA)—Level of Repair Analysis (LORA) is a logical procedure for identifying the cost of various maintenance alternatives for a specific item of hardware or software. LORA considers such issues as spares, facilities, staff, and support equipment. LORA is also known as Repair Level Analysis.

LORA is particularly useful in those cases where there is a direct hardware/software interface, such as the need for software loading after a hardware repair. LORA is described in [DEFS0060-0] and [MILPRF49506].

6.9.5 OBSOLESCENCE MANAGEMENT—Software-intensive systems are prone to obsolescence issues, particularly in those cases where the systems have a long operational life, or when the support concept considers the possibility of extending the operational life of the system or its associated support elements. Such obsolescence may arise due to factors such as hardware upgrades, changes in the operational or support environment, and enhancements to system functionality, or use of non-standard languages.

Obsolescence of component computer hardware resulting in new hardware may require adaptive changes to associated software. This can be a very costly issue if the support aspects are not considered during the design phase to ensure isolation of hardware dependencies in the software and seamless and cost-effective transition to the new hardware when the obsolescence occurs. The impact of obsolescence on both custom and COTS software should be analyzed.

Obsolescence Management ensures that obsolescence is treated as a support criterion, and thus considered as part of the support concept throughout the software life cycle. Obsolescence Management is described in more detail in [BS7000-5].

7. Notes

7.1 Marginal Indicia—The change bar (l) located in the left margin is for the convenience of the user in locating areas where revisions have been made to the previous issue of the report. An (R) symbol to the left of the document title indicates a complete revision of the report.

PREPARED BY THE SAE G-11 RELIABILITY, MAINTAINABILITY, SUPPORTABILITY AND LOGISTICS (RMSL) SOFTWARE COMMITTEE

REAFFIRMED BY THE SAE G-11 RMSL SOFTWARE COMMITTEE

APPENDIX A

GLOSSARY OF ACRONYMS AND TERMS

A.1 Acronyms

AFOTEC	Air Force Operational Test and Evaluation Center
AIR	Aerospace Information Report
ARP	Aerospace Recommended Practice
COCOMO	COnstructive COst MOdel
COTS	Commercial Off-The-Shelf
DRACAS	Data Reporting, Analysis, and Corrective Action System
EPI	Engineering Process Improvement
EPM	Equipment Program Management
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Modes, Effects and Criticality Analysis
FRACAS	Failure Reporting, Analysis, and Corrective Action System
IBR	Interim Block Release
IEC	International Electrotechnical Committee
IEEE	Institute of Electrical and Electronic Engineers
ISO	International Standards Organization
JA	Two character code for SAE ground vehicle (J) and aerospace (A) standards and guidelines
LORA	Level of Repair Analysis
LRI	Line Replaceable Item
LSA	Logistic Support Analysis
LSAR	Logistic Support Analysis Record
MOD	Ministry of Defence (United Kingdom)
NBR	Normal Block Release
OFP	Operational Flight Program
OTS	Off-The-Shelf
PR/CR	Problem Report / Change Request
RCTA	Requirements and Technical Concepts for Aviation, Inc.
RDIT	Replication, Distribution, Installation and Training
REVIC	Revised Intermediate COCOMO
RMSL	Reliability, Maintainability, Supportability, Logistics
SAE	Society of Automotive Engineers
SASET	Software Architecture Sizing and Estimating Tool
SCR	Software Change Request
SEER-SEM	System Evaluation and Estimation of Resources – Software Estimation Model
SEI	Software Engineering Institute
SEP	System Executive Planning
SEPO	Software Engineering Process Office
SLIM	Software Life Cycle Methodology
SRV	Service Repair Vehicle
USC	University of Southern California

A.2 Key Terms

A.2.1 Customer—An organization that procures a system, software product or software service from a supplier.

NOTE—The acquirer could be one of the following: buyer, owner, user, and purchaser. Equivalent with ISO/IEC 12207 definition of Acquirer.

A.2.2 Change Request Profile—Each software product change request can be described by a set of attributes across 3 dimensions: complexity (hi, med, lo), type (corrective, perfective, adaptive), and priority (normal, urgent, emergency). The Change Request Profile for a software product comprises all the expected change request attributes for a given period. For example, from the change request profile it should be possible to identify that over the period there are expected to be, say, 3 corrective changes of high complexity with 2 having normal priority and one urgent priority.

A.2.3 Failure, Fault, Mistake, Error: failure—The inability of a system or component to perform its required functions within specified performance requirements; fault—manifestation of a mistake, when executed may cause a failure; mistake—an incorrect human action; error—the affect of a failure.

A.2.4 Failure Effects, Modes and Criticality Analysis—A proactive approach used for determining the potential failure modes of a system/equipment (including software), all likely ways in which a component or equipment can fail, causes for each failure mode, and effects/criticality of each failure mode.

A.2.5 Failure Reporting and Corrective Action System—A set of processes, procedures, and tools for reporting, reviewing, analyzing, correcting, and storing information about system/software failures.

A.2.6 Integrated Logistic Support—A disciplined, unified, and iterative approach to the management and technical activities necessary to 1) integrate support considerations into system and equipment design, 2) develop support requirements that are related consistently to readiness objectives, to design, and to each other, 3) acquire the required support, and 4) provide the required support during the operational phase at minimum cost.

A.2.7 Logistics Management Support—The software support activities related to the 'bridge' between the software support services for operational use and for post-delivery modification support. Activities may include help desk management, problem reporting and corrective action coordination, distribution of releases to field sites, management of configuration information concerning software releases in the field, and network communications among field sites and modification support sites.

A.2.8 Logistic Support Analysis—The selective application of scientific and engineering efforts undertaken during the acquisition [and sustainment] process, to assist in 1) causing support considerations to influence design, 2) defining support requirements that are related optimally to design and to each other, 3) acquiring the required support, and 4) providing the required support during the operational phase at minimum cost.

A.2.9 Maintainability—The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. Also, a set of attributes that bear on the effort needed to make specified modifications.

A.2.10 Maintenance—the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment.[1]

1. Software maintenance as defined above is essentially the same as software modification support. Software maintenance and the maintenance process have not always been interpreted as defined in this appendix.

A.2.11 Modification Support—The software support activities of change analysis, implementation, test and release of software products. Changes may be termed corrective, perfective and adaptive, and may also embrace modifications that are designed to prevent foreseeable future software operating problems.

A.2.12 Operational Support—The software support activities related to the day-to-day operation of software by the software user. Activities may include installation, configuration, data preparation and loading/unloading, backup, recovery, failure reporting and training.

A.2.13 Supportability Characteristics—Attributes of software processes (e.g., development, transition, operation, support), products (e.g., documentation, source code, test suites), and environment resources (e.g., facilities, support systems, support personnel) that enhance the capability to support software.

A.2.14 Support Concept—The derived set of information that describes the strategy, primary factors, and performance requirements for one or more software products across the three dimensions of support classes, support functions, and support profile.

A.2.15 Support Classes—Cover the main quality characteristics of the major factors affecting software supportability. In principle, three factor classes are identified:

a. *Processes*—The inherent quality characteristics of the support processes that affect the capability to accomplish the required support functions;
b. *Product*—The inherent quality characteristics that are designed into the software product that affect the capability to accomplish the required support functions; and,
c. *Environment*—The inherent quality characteristics of personnel resources, support systems, and physical facilities that affect the capability to accomplish the required support functions.

A.2.16 Support Functions—Refers to the different activities required to be carried out during the software's operational use. Within this approach, software has three distinct, but closely linked, functional support areas: Operational Support, Logistics Management Support, and Modification Support.

A.2.17 Support Profile—Covers the overall understanding of how software support should be addressed. The Support Profile consists of three distinct aspects:

a. *Support Level*—The different (generic) sites where support will be carried out, their capabilities, resources and overall role in the support process.
b. Support Agent—The generic roles taken by organizations and groups of people that are involved in software support, and the responsibilities and information they require.
c. Support Scenarios—The individual tasks required to carry out all support processes, their interrelationships and sequencing and the people, organizations, resources, and infrastructure required to carry them out.

A.2.18 Supportability—A set of attributes of software design, the associated development tools and methods, and the support environment infrastructure that enable the software support activities to be accomplished.

A.2.19 Supportability Analysis—See Logistic Support Analysis.

APPENDIX B

SOFTWARE SUPPORTABILITY PLAN AND CASE TEMPLATES

B.1 Example Software Supportability Plan Template

1.0	*PROGRAM PLANNING AND CONTROL*
1.1	*Development of an Early Software Supportability Strategy*
1.2	*Software Supportability Plan and Procedures Manual*
1.3	*Program and Design Reviews*
2.0	*MISSION, DEVELOPMENT & SUPPORT SYSTEMS DEFINITION*
2.1	*Use Study*
2.2	*Support Environment Standardization*
2.3	*Comparative Analysis*
2.4	*Technological Opportunities*
2.5	*Support & Supportability Related Design Factors*
2.6	*Integration of ADP Systems*
3.0	*PREPARATION & EVALUATION OF ALTERNATIVES*
3.1	*Functional Requirements*
3.2	*Non-Functional Requirements*
3.3	*Support System Alternatives*
3.4	*Evaluation of Alternatives & Trade-Off Analysis*
4.0	*DETERMINATION OF SOFTWARE SUPPORT REQUIREMENTS*
4.1	*Operational Software Support Task Analysis*
4.2	*Software Exception/Problem Support Analysis (FRACAS)*
4.3	*Logistics Management Support Task Analysis*
4.4	*Software Modification Support Analysis*
4.5	*Data Support Task Analysis*
4.6	*Software Transition Analysis*
4.7	*Post-Deployment Software Support Analysis*
5.0	*SOFTWARE SUPPORTABILITY ASSESSMENT*
5.1	*Software Operational Supportability Assessment*
5.2	*Problem Reaction Assessment*
5.3	*Software Modification Supportability Assessment*
5.4	*Logistics Management Assessment*
5.5	*Production of the Supportability Case*
5.6	*Lessons Learned & Recommendations for Future Projects*
6.0	*IN-SERVICE SOFTWARE SUPPORTABILITY*
6.1	*Support Plan Evolution*
6.2	*Supportability Analysis During Support*
6.3	*Support Case Evolution*

FIGURE B1—EXAMPLE SOFTWARE SUPPORTABILITY PLAN TEMPLATE

Alternatively, the Software Supportability Plan could be outlined as follows:

1. MANAGING THE SOFTWARE SUPPORTABILITY PROGRAM ACTIVITIES
   - *a. Define purpose, scope of plan and program, reliability goals and objectives*
   - *b. Nomenclature and project references*
   - *c. Program management functions, responsibility, authority, interaction between system engineering, integrated logistic support, and software supportability activities*
   - *d. Resources needed*
     - *i. Personnel, skills*
     - *ii. Equipment*
     - *iii. Schedule*
   - *e. Training*
   - *f. Subcontract Management*
   - *g. Plan approval and maintenance*
   - *h. Customer interaction/involvement*
2. *PERFORMING SOFTWARE SUPPORTABILITY PROGRAM ACTIVITIES*
   - *a. Define lifecycle model and methodology, interaction with system engineering*
   - *b. Identify specific static and dynamic analyses to be performed throughout lifecycle (are you on target for meeting your supportability goals and objectives)*
   - *c. Analysis of pre-existing software*
   - *d. Transition to operational environment*
   - *e. Training end-users, operations and support staff*
   - *f. Decommissioning*
3. *DOCUMENTING SOFTWARE SUPPORTABILITY PROGRAM ACTIVITIES*
   - *a. Lifecycle artifacts*
   - *b. Software support concept*
   - *c. Software supportability case file*

FIGURE B2—ALTERNATE SOFTWARE SUPPORTABILITY PLAN

B.2 Example Software Supportability Case Template

1. *SOFTWARE SUPPORTABILITY GOALS AND OBJECTIVES*
   - 1.1 *What they are: software support significant items*
   - 1.2 *How were they derived, apportioned to software*
   - 1.3 *Relation to system supportability goals*
   - 1.4 *Regulatory and/or contractual requirements*
   - 1.5 *Agreed upon verification and validation criteria*
2. *ASSUMPTIONS*
   - 2.1 *Support concept tailoring and strategy*
   - 2.2 *Customer/supplier interfaces*
   - 2.3 *Development vs. OTS concerns*
3. *EVIDENCE*
   - 3.1 *Supportability analysis activities that demonstrate achievement of software supportability goals and objectives*
     - 3.1.1 *Pre-development phase*
     - 3.1.2 *Development phase*
     - 3.1.3 *In-service phase*
   - 3.2 *Product, process, environment characteristics that demonstrate achievement of software supportability goals and objectives*
     - 3.2.1 *Pre-development phase*
     - 3.2.2 *Development phase*
     - 3.2.3 *In-service phase*
4. *CONCLUSION/RECOMMENDATION*
5. *SUPPORTABILITY ANALYSIS RECORDS*

FIGURE B3—EXAMPLE SOFTWARE SUPPORTABILITY CASE TEMPLATE

APPENDIX C

SOFTWARE LOGISTICS SUPPORT ACTIVITY TASKS

A description of the set of tasks (including the program planning tasks that establish a general software supportability strategy and a plan to accomplish that strategy) is outlined below with possible inputs and outputs. A more complete description of similar tasks within a military application context is provided in [DEFS0060-2] and [DEFS0060-3]. A process flow for software supportability analysis is presented in [CASA-SAS].

C.1 Program Planning and Control—The purpose of these tasks is to identify life cycle programmatic plans and controls that will be used to address system/software supportability and support.

C.1.1 Development of an Early Software Supportability Strategy—This task should develop a strategy for a software supportability program based on what supportability principles and policies are to be followed in deriving a software support concept. Using available early data, the strategy should determine the method for selecting necessary software support tasks, identify known support organizations required to undertake selected tasks, and estimate the cost effectiveness of each individual task, the goals to be met, and the criteria to be used to assess achievement of those goals. The strategy should also consider the requirement for tools to model and determine support costs.

a. *Inputs*—Contract, project definition, historical data.
b. *Outputs*—Software Supportability Strategy.

C.1.2 Software Supportability Plan and Procedures Manual—A Software Supportability Plan has to be defined which details and integrates all tasks to be carried out, management and organizational responsibilities, provides detailed planning and schedules, and defines specific elements such as data to be collected, as well as the processes to be carried out to ensure software supportability. A Software Supportability Plan will, in many cases, be a contractual document.

It is also convenient to develop a Procedures Manual, for the detailed definition of the processes that will be used for the actual analysis work, as a reference guide.

a. *Inputs*—Contract, Software Support Strategy.
b. *Outputs*—Software Supportability Plan, Procedures Manual.

C.1.3 Program and Design Review—It is necessary to plan and carry out formal reviews of software supportability related design information and software supportability process outputs (Data Repository information, reports, and so forth) in a timely and controlled manner. The reviews should assure both management and the customer that the supportability program is proceeding as planned and that software supportability requirements will be achieved. It is also necessary to ensure the integration of such reviews within the overall design and global supportability reviews.

a. *Inputs*—Software Supportability Plan.
b. *Outputs*—Review Procedures, Management tracking of milestones.

C.2 Mission, Development, and Support Systems Definition—The purpose of these tasks is to determine how software support is defined in relation to existing systems, the target system, and future system evolution.

a. *Inputs*—Contract, project definition, historical data, customer requirements, existing system support data, technology direction.
b. *Outputs*—Support requirements and top-level support concept: support scenarios, potential support factors, support candidates and risks, supportability data.

C.2.1 Use Study—Prior to any software supportability assessment, it is essential to know both the intended usage of the end product and the environment under which support for this product will be carried out. The customer, together with the contractor, has to define as clear as possible the environment under which the software will operate and where support should be carried out, within organizational constraints, human capabilities and limitations, required turn-round times or budgetary limitations, strategic considerations or security aspects, as well as operational impact of potential problems. Support scenarios that capture the sequence of existing or expected support actions can provide much of this understanding. A use study is key to the definition of the overall support concept, as the software supportability program will attempt to comply as much with this global framework in the most cost-effective way.

a. *Inputs*—Customer Requirements.
b. *Outputs*—Use Study results and updates due to life cycle modifications.

C.2.2 Support Environment Standardization—This task establishes supportability and supportability related design constraints for the different software support environments. These constraints will be based on existing and planned computing infrastructure or other support resources, tools, relative cost, staff effort, type of staff, readiness and support policy considerations.

a. *Inputs*—Mandatory design constraints or requirements, information from existing or planned support environments.
b. *Outputs*—Qualitative or quantitative tool or equipment standardization guidelines or constraints, based on readiness, relative cost, or other supportability considerations, associated risks.

C.2.3 Comparative Analysis—This task will assess similar projects, comparing information about their successes and failures, for an analogy with the current one, in order to define a skeleton support concept, evaluate supportability and cost factors and examine previously used methodologies and plans.

a. *Inputs*—Historical data, past projects, published experiences.
b. *Outputs*—Skeleton support concept, potential supportability factors.

C.2.4 Technological Opportunities—The evaluation for technological opportunities such as new processors, operating systems, and new CASE tools is a necessary step in order to improve both design and support aspects, as well as to reduce life cycle cost and prolong the useful life of the fielded product. This analysis, however, should not forget the impact and risks that are always associated to new technologies.

a. *Inputs*—New technologies, product information.
b. *Outputs*—Design requirements, migration plans.

C.2.5 Support and Supportability Related Design Factors—This task establishes supportability and supportability related design guidelines for the software to be supported. These factors will be based on existing and planned computing application and support environments, documentation, methods, relative cost, software product aspects, technology, expected change, readiness and support policy considerations.

a. *Inputs*—Results from Task C.2.2, mandatory design constraints or requirements, information from application domain, product factors.
b. *Outputs*—Qualitative and quantitative software supportability objectives for the support candidates and associated environment, as well as risks associated to those objectives, thresholds and constraints.

C.3 Preparation and Evaluation of Alternatives—The purpose of these tasks is to analyze functional and non-functional support requirements for all support functions against all potential alternative systems and support concepts.

a. *Inputs*—Customer requirements, alternative systems and support concepts, operational needs, support scenarios, trade-off tools and methods.
b. *Outputs*—Selected support system alternative, agreed to functional and non-functional support requirements, detailed cost and effort estimations, estimated resources, estimated performance response times.

C.3.1 Functional Requirements—This task lists and documents all functional requirements associated with the operational use and/or support of the software for all software support functions, and against all potential alternative systems and possible support concepts, in order to ensure full compliance.

a. *Inputs*—Customer requirements, alternative systems.
b. *Outputs*—Design aims, functional requirements.

C.3.2 Non-Functional Requirements—This task addresses the identification and documentation of all non-functional requirements associated to the product and different support functions. Such non-functional requirements (e.g., security, health regulations) might have a great impact on both design and support and thus on potential support solutions.

a. *Inputs*—Customer requirements, operational needs, support scenarios.
b. *Outputs*—Design aims, non-functional requirements.

C.3.3 Support System Alternatives—The definition of different support alternatives is key to the optimization of software support. These support system alternatives can be modeled by means of support scenarios, which will be an initial outline of different support concepts, including not only the process, but also the necessary support infrastructure, support level, agents, staff effort, and so forth.

a. *Inputs*—Support requirements, Use Study.
b. *Outputs*—Set of detailed support scenarios.

C.3.4 Evaluation of Alternatives and Trade-Off Analysis—This task carries out a comparison between the previously identified support system alternatives, identifying all necessary support infrastructure, as well as potential risks and pitfalls, such as availability of the support systems upon entry into service. This task also ensures that such support infrastructure is planned for and properly considered during design.

a. *Inputs*—Use Study, Support System Alternatives, Trade-Off tools and methodologies.
b. *Outputs*—Detailed effort and cost estimations, required resources, turn-round times.

C.4 Determination of Software Support Requirements—The purpose of these tasks is to analyze functional and non-functional support requirements for all support functions against all potential alternative systems and support concepts.

a. *Inputs*—Customer requirements, functional and non-functional support requirements, selected systems and preliminary support concept.
b. *Outputs*—Quantified personnel, skills, and training, tools, and response times for all system/software support tasks. Completed Software Support Concept. Preliminary Software Support Case of evidence that the software is supportable in accordance with the Concept. Transition plan and recommendations for continued Software Support Analysis through deployment of software for operational use.

C.4.1 Operational Software Support Task Analysis—This type of analysis covers the different aspects of the operational support function such as loading, installation, configuration and operation. It must be closely coordinated with similar analysis carried out on hardware (e.g., LSA), in order to provide compatible support characteristics and methods, as well as to share as much as possible support resources. Techniques like Reliability Centered Maintenance or LORA are likely to be used also for this analysis. The results of this analysis are usually recorded in a Logistic Support Analysis Record (LSAR).

a. *Inputs*—FMECA or Support Scenario triggers, Use Study.
b. *Outputs*—Identification of Support resources and requirements, LSAR.

C.4.2 Software Exception/Problem Support Analysis (FRACAS)—This type of analysis covers those cases where, due to operator error, design flaws or exceptional circumstances (e.g., power failure, flood, earthquake) the software is unable to carry out its intended functions. The analysis considers the consequences of such failures, on human safety, material or economical damages or similar, and determines design alternatives, mitigation actions and necessary resources. Note that the outputs of this analysis also feed into the software modification analysis.

a. *Inputs*—FMECA or Support Scenario Triggers, Use Study.
b. *Outputs*—Problem Reporting and Corrective Action Process, Contingency Plans.

C.4.3 Logistics Management Support Task Analysis—This type of analysis covers all the different aspects of the logistics management support function, such as replication of software, distribution of software and associated documentation, delta training, and help desk. It should be kept in mind that this aspect is the bridge between software operation and software modification, so it is important that a coherent concept is developed.

a. *Inputs*—Product information, Use Study, Support System Alternatives.
b. *Outputs*—Software Logistics Process Plan, Identification of Resources.

C.4.4 Software Modification Support Analysis—This type of analysis covers the different aspects of software modification, due to corrective, perfective or adaptive maintenance. This analysis covers the whole modification cycle, from problem investigation to software release, including determination of typical effort and time scales and necessary support resources, and provides feedback to design on all those elements that can improve response times or reduce support efforts and costs.

a. *Inputs*—Product information, Use Study, Support System Alternatives.
b. *Outputs*—Software Modification Process Plan, Identification of Resources.

C.4.5 Data Support Task Analysis—This type of analysis covers the different aspects associated to data, in a similar way as for software. It considers not only the use of data by different software (and hence the impact of modifying its structure), but also all aspects such as relocation of those data, preparation, validation if necessary, and possibly post-production analysis.

a. *Inputs*—Product information, Use study.
b. *Outputs*—Data Support Plan, Identification of Resources.

C.4.6 Software Transition Analysis—This analysis considers the existing support infrastructure and contrasts it to the overall Support Concept; taking into account project plans and schedules, in order to document a smooth transition for the initial deployment, usage and support. This analysis considers interim solutions, staffing plans, training curves, support equipment acquisition and similar aspects as to ensure that the overall Support Concept will be implemented at a specific date after the first product is released into operation.

a. *Inputs*—Use Study, Project Plans, Support Systems “current” and “future”, Support Concept.
b. *Outputs*—Software Deployment or Transition Plan.

C.4.7 Post-Production Software Support Analysis—This type of analysis covers all aspects related to the software once it has been fielded and is fully operational. In particular, it has to assess the different problems that might arise regarding support issues (e.g., key suppliers discontinuing support, new application domains or sites) or product or support environment obsolescence.

a. *Inputs*—Use Study, defined Support Concept.
b. *Outputs*—Contingency Plans, Escrow Agreements, and Pre-Planned Product Improvement.

C.5 Software Supportability Assessment—The purpose of these tasks is to provide customer and supplier assurance that the projected software support capability will be adequate. Assess the software support capability in accordance with the Software Support Plan, Software Support Concept, and Software Support Case. Direct measurement, demonstrations, and evaluation evidence may be part of the assessment.

a. *Inputs*—Customer requirements, functional and non-functional support requirements, Support Plan, Support Concept, preliminary Support Case.
b. *Outputs*—Assessment measurements and results.

C.5.1 Software Operational Supportability Assessment—This assessment covers a final review of the support concept, in order to test and validate the assumptions, processes and resources required for software operation, possibly in conjunction with a formal OT&E process for the overall system in which the software is embedded. In certain cases, this assessment might be carried out by a different organization than the one that developed the Support Concept.

a. *Inputs*—Operational Support Concept.
b. *Outputs*—Operational Supportability Assessment Report.

C.5.2 Problem Reaction Assessment—In those cases where a system shutdown or software failure might have safety implications or cause important economic damage, a walkthrough or independent audit of the Problem Reaction part of the support concept might be desirable, in order to verify/validate the contingency plans that were previously established as part of the Problem Reaction Analysis.

a. *Inputs*—Contingency Plans, FRACAS.
b. *Outputs*—Problem Reaction Assessment Report.

C.5.3 Software Modification Supportability Assessment—This assessment covers a final review of the modification support concept, including a final verification of the availability of required support resources, in order to test and validate the assumptions, processes and resources required for software modification.

a. *Inputs*—Software Modification Support Concept.
b. *Outputs*—Software Modification Capability Assessment Report.

C.5.4 Logistics Management Assessment—This assessment covers a final review of the logistics management support concept, including a final verification of the availability of required support resources, in order to test and validate the assumptions, processes and resources required for software logistics, including RDIT, possibly as part of the initial deployment activity.

a. *Inputs*—Logistics Management Support Concept.
b. *Outputs*—Software Logistics Capability Assessment Report.

C.5.5 Production of the Supportability Case—This task covers the production of the supportability case, including the recompilation of all necessary evidence and demonstration of achievement of the software supportability requirements in accordance with the stated in the software supportability case.

a. *Inputs*—Assessment Reports, Supportability Project information, data repositories.
b. *Outputs*—Supportability Case.

C.5.6 Lessons Learned and Recommendations for Future Projects—A final task to be carried out is a recompilation of all the project history, its proper archiving and final assessment. In particular, a project summary detailing the major problems encountered, as well as the lessons learned should be produced. Key elements relating to the project management, supportability study, support criteria and even software design should be highlighted in the form of recommendations for future projects, so as to preserve the knowledge acquired during the present one. Lessons Learned assist in the Comparative Analysis task C.2.3 of other projects.

a. *Inputs*—Project history, project documentation.
b. *Outputs*—Project Summary and Lessons Learned.

C.6 In-Service Software Supportability—During the in-service phase, a continuous re-evaluation of the Support Concept is required, in order to sustain the operation, modification or logistics management of the supported product. Metrics should be collected about the service quality, different support costs, encountered problems, etc., but supportability should be also assessed against new factors such as new technologies or emerging problems such as gradual obsolescence. Towards the end of the software life cycle, a retirement plan and/or transition plan to a new system should be prepared.

a. *Inputs*—"Current" Support Concept, Metrics, new technologies, emerging support problems.
b. *Outputs*—Updated Support Concept, Plan and Case, improved support infrastructure, Retirement Plan, Transition Plan.

APPENDIX D

SOFTWARE SUPPORT SCENARIO EXAMPLES

D.1 Example Operator Problem Reporting Support Scenario—The following example demonstrates the simplified modeling of a software support scenario, in this case an operator problem reporting through resolution.

D.1.1 Scenario Case: Operator Problem Reporting Scenario—The operator problem reporting example scenario is initiated by the system operator observing a system failure. This failure support trigger event typically results in an operator report followed by a problem/change request that requires a correction to software, enhancement/adaptation to the software, or assistance in resolving some problem through operational services or workaround procedures. The problem/change resolution process handles the initial aspects of the software support. Changes to the software are controlled by the modification activities of analysis, implementation, test, and release. The release management activities ensure any software-specific changes are properly integrated into an appropriate software release. Control of release configuration, software versions, and inventory/logistics management of software releases is through configuration management activities. Various field, intermediate, and depot support activities are illustrated in this example.

Name:

Operator Problem Reporting Support Scenario Case

Pre-Conditions:

The system operator observes a system problem that appears to be related to an application software capability. This software problem is assumed to require the full level of system support, including activities such as system problem report generation, change request generation, integration of the change into a software block release, control of the software version and logistics control of the release, distribution of the release to the field, and transition of the release into the specifically designated field equipment. Field, intermediate, and depot levels of support are illustrated in this example.

Steps:

1. Operator observes system failure.
2. Operator raises operator report to the field help desk.
3. Field supporter provides a quick help desk response to the field and studies the operator report, determines that a problem does exist, and raises a new problem report to the intermediate site supporter; problem report is logged into field log.
4. The intermediate site supporter analyzes the problem report, determines a change is needed that can not be done at the intermediate site, and provides a possible change request for the depot supporter; change request is logged into the intermediate log.
5. The depot supporter receives the problem report and suggested change request from the intermediate site and enters the problem report and change request into the PR/CR tracking system that initiates a modification cycle for the change request.
6. The depot supporter analyzes the change request, confirms the needed change, allocates the change to the next normal block release, and implements the change.
7. Upon completion of the normal block release, the release is distributed along with distribution/ installation instructions to the intermediate site supporter; the PR/CR status is updated to closed.
8. The intermediate site supporter distributes the software and appropriate documentation to the field supporter for installation; intermediate log is updated to indicate the change request and problem report are closed.
9. The field supporter installs the new software release, performs acceptance test procedures, and verifies operational capability per distribution/installation instructions; applicable operator report(s) closed.

Post-Conditions:

The system equipment affected by the failure has been updated with the appropriate software release. Problem report and change request information has been closed and information databases updated. Configuration management information has been updated and inventory logistics management information has been updated. Operator problem has been resolved.

Variations:

1. Step 1: System failure is able to be fixed through operator workaround; operator reports failure and workaround solution; no other actions required.
2. Step 3: Field supporter provides a quick help desk workaround response to the field that solves the operator problem; operator report and workaround are logged; no other actions required.
3. Step 6: The depot supporter analysis provides a quick workaround solution that is relayed to the field through the intermediate supporter; operator problem is resolved; problem report and change request are closed with the workaround solution; no other actions required.
4. Step 6: After analysis of the problem, some other possible alternatives may occur: a solution is not feasible, change request is not implemented; operator error has occurred and no software changes are needed, operator procedures are updated as the resolution; change is determined to be emergency and emergency block release process is invoked.
5. Steps 7,8,9: emergency release distribution procedure is invoked that bipasses the intermediate site and provides distribution and installation support directly to the field; intermediate site is notified; change request is put back into the release cycle to ensure the next normal block release includes the change.

D.1.2 Scenario Sequence Diagram—The concept of sequencing each of the steps and providing some further information concerning the agents, activities, and interactions for each step/node is illustrated in Figure D-1. There are many possible representations for this kind of diagram. Notations for the sequence diagram nodes are:

Node Number	Function “Fn”	Support Function Name
	Level “Ll”	Support Level Description
	Agent “Ag”	Support Agent Type

SUPPORT SCENARIO MODEL - OPERATOR PROBLEM REPORTING

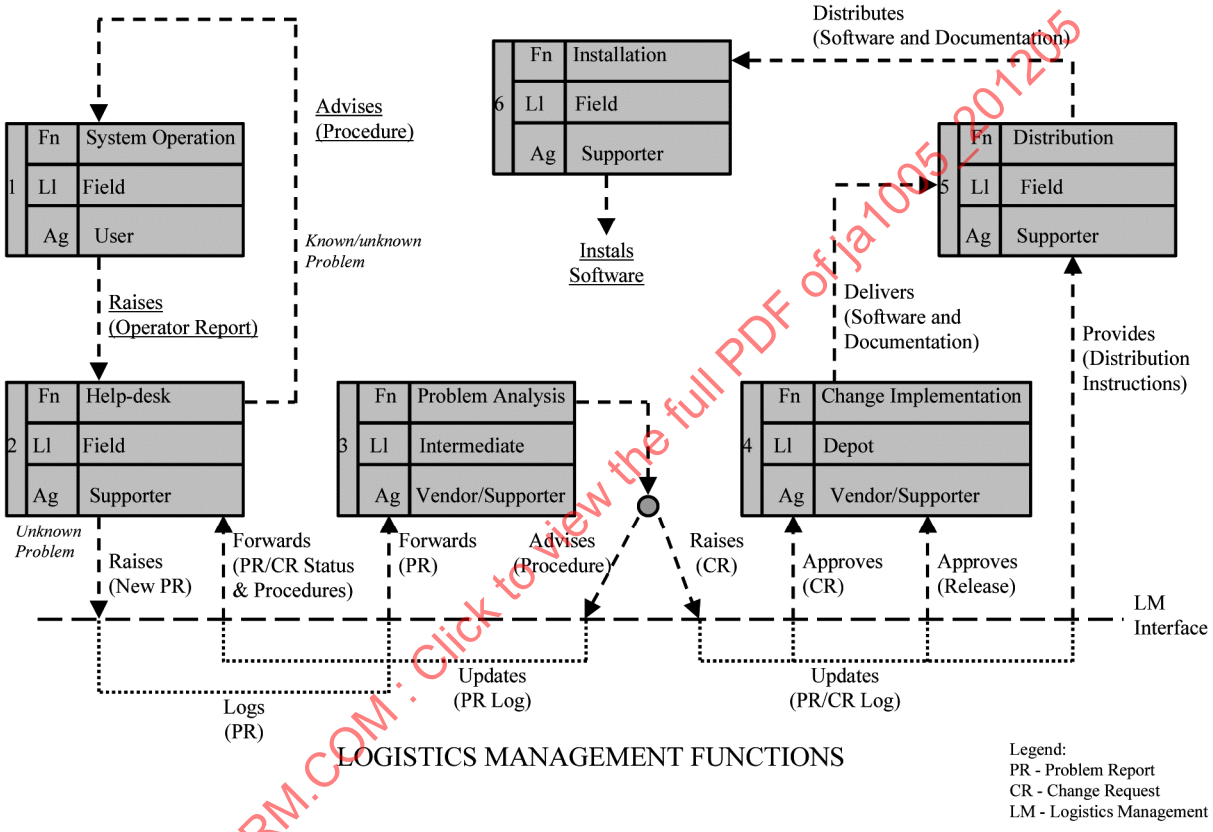


FIGURE D1—SAMPLE SUPPORT SCENARIO MODEL—OPERATOR PROBLEM REPORTING

D.1.3 Step/Node Chart—An example step/node chart that describes the inputs, outputs, and procedures for each node is illustrated in Figures D2 through D5. Notes:

a. Logistics Management is treated as a ‘node,’ although not depicted as such in the support scenario model.
b. Nodes 4-6 are not expanded in this example.