A Three-tier Knowledge Management Scheme for C2 Software Engineering Support and Innovation

Richard D Corbin
Northrop Grumman
Defense Mission Systems
Bellevue, NE 68005
Corbinr@stratcom.mil

Christopher B. Dunbar
FGM Inc.,
7126 N 121st Street
Omaha, NE 68142
dunbar@fgm.com

Qiuming Zhu
College of Information Science and Technology
University of Nebraska
Omaha, NE 68182
zhuq@unomaha.edu

A Three-tier Knowledge Management Scheme for C2 Software Engineering Support and Innovation

Richard D Corbin
Northrop Grumman
Defense Mission Systems
Bellevue, NE 68005
Corbinr@stratcom.mil

Christopher B. Dunbar
FGM Inc.,
7126 N 121st Street
Omaha, NE 68142
dunbar@fgm.com

Qiuming Zhu
College of IS&T
University of Nebraska
Omaha, NE 68182
zhuq@unomaha.edu

Abstract

To ensure smooth and successful transition of software innovations to C2 systems, it is critical to maintain proper levels of knowledge about the system configuration, the operational environment, and the technology in both existing and new systems. We present a three-tier knowledge management scheme through a systematic planning of actions spanning the software transition process from conceptual exploration to prototype development, experimentation, and product evaluation levels. The three-tier knowledge management scheme is an integrated effort for bridging the development and operation communities, maintaining stability and minimal impact to the operational performance of C2 systems, while swiftly adapting to technology innovations. A current effort, Command and Control Software Engineering and Support - C2SES, underway at United States Strategic Command (USSTRATCOM) will serve as a real-world test case for the approach. Knowledge management in the C2SES involves engaging a mix of technical expertise and qualifications concerning the use of an application by the USSTRATCOM user community. The targeted development combines resources of academic research laboratories, software industrial technology centers, and military software integration laboratories for introducing innovative solutions to C2 operations.

Key Words: Knowledge Management, C2 Software Engineering, Technology Innovation, Human and Operational Systems, Software System Support, Net-Centric Applications

I. Introduction

I.1. The problem

This paper addresses the problems and issues of knowledge management (KM) in a C2 software engineering support process. The problem is discussed in the context of how to maintain proper levels of knowledge in the processes of developing and transiting technological innovations into C2 software systems. The innovation processes typically include

(1) Insertion of new technology (e.g., advanced data dissemination, fusion, and knowledge discovery components and decision support agents) into existing systems to enhance the C2 system's operational capabilities,
(2) Transformation of legacy systems into net-centric integrative systems, and

(3) Installation of new systems to replace outdated systems, add novel capabilities to C2 systems, and support closing the gaps in required capabilities.

Relevant issues of knowledge management to be addressed in this paper include:

(1) Knowledge acquisition – how to acquire and retain knowledge from existing software development teams, personnel, and subject matter experts (SME),
(2) Knowledge maintenance – how to keep the knowledge up-to-date and validated over time, and avoiding knowledge evaporation and obsolesce,
(3) Knowledge sharing - how knowledge is disseminated/communicated to the team of collaborators and shared by all stakeholders, and
(4) Knowledge enhancement – how to turn weak knowledge into strong knowledge that is useful for the C2 software innovation and engineering support tasks.

It is known that "Knowledge management is an approach to discovering, capturing, and reusing both tacit (in people's heads) and explicit (digital or paper based) knowledge as well as the cultural and technological means of enabling the knowledge management process to be successful [Rec05]." However, what is "Knowledge" remains to be an issue in some people's mind, especially referring to specific domains and situations. Many knowledge management practitioners and researchers considered *information* and *knowledge* as synonymous constructs. In this perspective, both these constructs can be expressed in the computational rule based logic as well as in the form of data inputs and data outputs that trigger pre-defined and pre-determined actions in pre-programmed modes [Wi03]. Unfortunately, this model is faulted with significant inaccuracy and limitations.

Information systems researchers [Chu71, MM73, Ma97] have discussed limitations of this model, particularly for environments characterized by uncertainty and radical change. Churchman [Chu71], who developed five archetypal models of inquiring systems in an effort to expand the field of management information systems along a philosophical path, has emphasized that: "To conceive knowledge as a collection of information seems to rob the concept of all of its life... Knowledge resides in the user and not in the collection." Similarly, Nonaka and Takeuchi [NT95] had proposed the conceptualization of knowledge as justified belief in their argument that, "knowledge, unlike information, is about beliefs and commitment." On a complementary note, Davenport and Prusak [DP98] have defined knowledge as deriving from minds at work: "Knowledge is a fluid mix of framed experience, values, contextual information, and expert insight that provides a framework for evaluating and incorporating new experiences and information. It originates in the minds of the knower. In organizations, it often becomes embedded not only in documents or repositories but also in organizational routines, processes, practices, and norms."

From the information technological point of view, knowledge is an entity differentiated from the information object in that there is an element of expert review and distillation where knowledge is concerned (See Figure 1 for a denotation of data-information-knowledge hierarchy). This view emphasizes that

1. Knowledge results from the fusion of key elements of information which characterize the problem space and includes explicit information (e.g. position of forces, geography, and weather) that requires little interpretation and can be communicated quickly and easily;

2. Knowledge yields predictive ability based upon interpretations that in consequence is based upon experience and a priori knowledge that includes tacit information (e.g., capabilities and tactics of an adversary, local customs, intents.) from which supporting facts can be easily transferred while the underlying organizing logic can seldom be transferred quickly and easily.

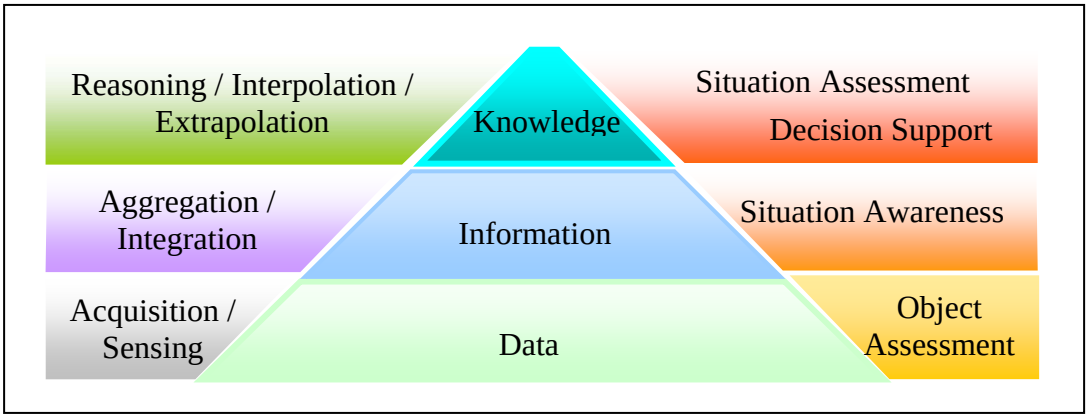

Figure 1. The Data-Information-Knowledge Hierarchy

Thus, the difference between information and knowledge is the degree of understanding – a functionalism of expertise and experience. It is seen that information results from the collection and assembly of "facts (data)" while knowledge involves the human element. From this point of view, sharing information is easier because it involves the transmission of "facts" which require relatively little interpretation. Sharing knowledge is far more difficult due to the fact that reasoning must be conveyed. Knowledge builds upon the foundation established by information and is, by way of contrast, people-intensive. As an instance, one could think of the handling of the former (information) in an automated process of data aggregation, while the handling of the latter (knowledge) in a process of conceptual pattern discrimination.

I.2. The Significance

To render successful transition and capability improvement in C2 software innovation, it is critical to maintain proper levels of knowledge within and around the processes. These include the knowledge about the software systems, the operating environment, the development processes, and the technologies in both existing and new systems. The objectives of all the efforts of such tasks are to ensure the efficiency and reliability of the overall C2 systems, avoid un-mature insertion of technology, and unnecessary interruptions to the system's operations (e.g., minimizing software defect incidents).

Knowledge management for software technology innovation includes processes of knowledge discovery, capture, storage, retrieval, sharing, and understanding. It aims at facilitating knowledge flow and utilization across every phases of a software engineering process. To be more specific, knowledge management in C2 software engineering is important due to the following reasons:

(1) The software development environment is characterized by frequent technology changes, which calls for a continuous stream of new knowledge.

(2) The transition of software innovation into practical operations requires a relatively large effort in requirement specification, prototyping, design validation, coding, and integration of various components. The risk would be larger if incorrect assumptions or approaches were noticed at the later stages of implementation and test.
(3) The benefits from the software improvement are often intangible and hard to assess. Managers and software engineers need to assess the benefit and suitability to mission enhancement and effectiveness at proper levels and stages with necessary knowledge from both sides in hand.

In other words, knowledge management in software engineering support has the mission to enhance the stability of the software system transition, and ensure swift adaptation of new technology to the C2 operations.

In the remainder of this paper, we will discuss the coupling of knowledge management tasks with the software development, transition, and maintenance processes section II, where knowledge management appear to be a critical component in C2 software engineering support, innovation, and performance improvement. Section III deals with the major knowledge management issues in a typical software development and transition process. We present a three-tier knowledge management scheme through a systematic planning of actions spanning from conceptual exploration to prototype development, product evaluation, and process execution levels. Section IV presents a current C2 software engineering effort, the Command and Control Software Engineering and Support - C2SES, underway at United States Strategic Command (USSTRATCOM). The project will serve as a real-world test case for the three-tier knowledge management scheme and an integrated effort for bridging the technical and operational communities. We conclude the presentation with a summary of the scheme and approach in section V.

II. Knowledge Management in C2 Software Development and Innovation

II.1. Knowledge management issues in C2 software innovation

As we all understand that software engineering is a multi-phased knowledge intensive process. The tasks of maintaining appropriate levels of knowledge in the process consist of many facets that can be described as different sub-problems. Examples of these sub-problems include:

(1) Liquidity of knowledge - Knowledge moves with people. For example, when employees leave for other opportunities or for retirement, they carry away certain amount of knowledge (e.g., in the form of experiences) no matter how well the knowledge has been documented.
(2) Latency of knowledge - To gain knowledge requires time. For example, when new employees are hired and they need to get up to speed by acquiring technical and subject matter knowledge, which often takes some time.
(3) Lack of knowledge sharing protocol – The power of knowledge lies on its sharing. For example, domain experts need to pass their knowledge to team players. To do that they need to know exactly to where and to whom the knowledge should be passed.
(4) Loss of knowledge locations, traces, and links – It is known that knowledge must be properly kept in places that are easily accessible. However, it is often overlooked or

misplaced, especially when under complex conditions and heavy work load. For example, when an urgent software fix was called upon, it was not clearly documented with respect to who was involved in a previous software fix and how it was done (i.e., what/where is the authoritative knowledge base for a specific software fix?).

A common problem in knowledge management is the ability to easily and efficiently locate and access the right knowledge at the right time to solve a particular problem. The knowledge is probably already present in many forms and organizations, potentially in hardcopy or some softcopy data management systems. The problem is to identify where it is or who has it. Technology can help capture some of the information but it is not the ultimate answer. That is to say, technologies such as software tools alone cannot solve the knowledge management problems totally. To have an effective knowledge management process, it is necessary to adopt a systematic scheme for planning, stipulating, and distributing the tasks and activities comprehensively. A key concept in this scheme is to take knowledge management as an active, affective and dynamic component of the software engineering process. That is, consider knowledge management as an inseparable dimension in the whole software development, transition, and maintenance processes. This leads to a scheme known for taking of knowledge somehow as information in action [Ma97].

The representation of knowledge as *information in action* rather than static computerized representations is notable because of several reasons. The *active*, *affective*, and *dynamic* (*AAD*) representation of knowledge makes sense from a pragmatic perspective of knowledge management in software engineering processes. The *AAD* representation of knowledge is better aligned with theoretical representations of the knowledge construct beyond the domain of information technology management [Ma97]. It is *active* as knowledge is best understood in action - it is not the theory but the practice of theory that makes the difference. It is *affective* as it takes into consideration not only the cognitive and rational dimensions but also emotional dimensions of human decision-making. It is *dynamic* as it is based upon ongoing reinterpretation of data, information, and assumptions while proactively sensing how decision-making process should adjust to future possibilities of alignment and adaptation. From a pragmatic perspective, the dynamic representation of knowledge provides a more realistic construct in general, where human and social interactions are present while situating this construct more proximal to performance outcomes [Ma97].

II.2. Coupling of knowledge management and software engineering processes

While knowledge management and process engineering were being evolved in parallel practically, there was no serious effort to fuse them into a consistent, holistic architecture. For example, knowledge management programs over the past decade have focused on organizing employees into communities of practice and building repositories of “best” or proven practices. There was (and still is) a general lack of understanding of how valuable the coupling of software engineering processes and the knowledge management practices can be.

One of the problems for an effective knowledge management practice is to understand the variants in different software engineering processes and how best these processes can be integrated with a knowledge management approach. Engineering processes do not just exist as structured or unstructured. They fill a range in between the two extremes. For example,

software engineering processes are filled with methods and metrics (e.g., CMMI). However, how a specific software fix was accomplished also depends on who did it (a human factor). Knowledge management schemes need to work adaptively in the whole range of the situations. That is to say, knowledge management must be closely linked to a particular group of people and of processes.

In the context of incorporating engineering processes with knowledge management practice, let us consider specifically a C2 software innovation process, and take a look at what processes it consists of. Namely, at a properly abstracted level, these processes are

1. Exploration – Identifying novel ideas and promising techniques for improvement to existing system capability or a new capability for insertion to existing systems.
2. Evaluation – Putting together and acting on a plan for assessing the ideas and opportunities, comparing and assessing the cost, efficiency, and technical feasibility.
3. Execution – Placing the innovative idea into development and operation stage upon the outcomes of evaluation and planning stages using system engineering techniques.

Figure 2 below shows the multi-phased processes of software engineering and the involvement of knowledge management in the processes.

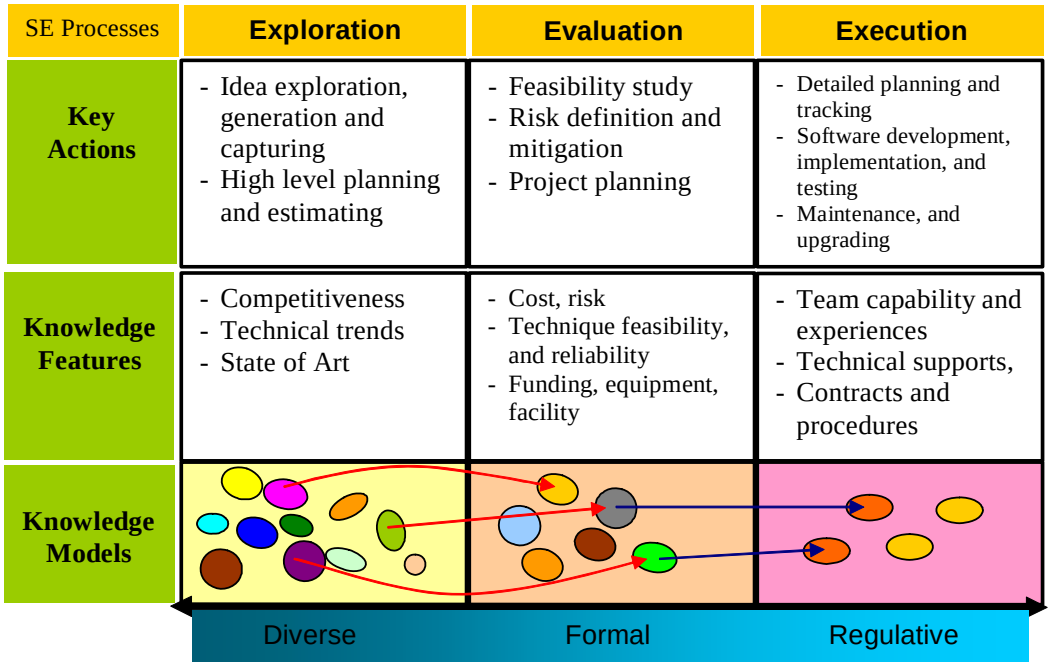

Figure 2 Knowledge management in a multi-phase C2 software innovation process.

II.3. A systematic plan of action

Driven by a growing understanding that knowledge is mostly intangible, difficult to hold on to, and usually a product of collective thought, knowledge management systems (KMS) become an interest to many research groups and organizations. KMS are often defined in terms of inputs such as data, information technology, best practices, etc., which by themselves may inadequately explain business performance outcomes. Often, moderating and intervening variables may play a significant role in skewing the simplistic relationships based upon correlation of the above inputs with business performance outcomes. Also, usefulness of such inputs and how they are

strategically deployed are important issues often left unquestioned as 'expected' performance outcomes are achieved, but the value of such performance outcomes gets eroded by the dynamic shifts in the software operating environments.

The dynamic, evolutionary approach of knowledge management calls for a system architecture that anticipates change and that fosters the systematic injection of upgraded systems, subsystems and components. One scheme of knowledge management in our C2 software engineering support is to establish an orderly combination of related parts and sub-systems, and an outline of knowledge organizations. We recognize that knowledge relevant in C2 software engineering and technology innovation includes

1. *Tangible (hard) knowledge*: e.g., (1) software products, (2) data repositories, (3) developing, testing, and maintenance standards and procedures, etc.;
2. *Intangible (soft) knowledge*: e.g., (1) intellects/subject matter experts, and war-fighters (2) system administrators, project managers, (3) software engineers, sub-contractors, collaborators, (4) operational environment and its variations, etc. "Knowledge does not simply exist – people create it." KMS can help people do better through its impact on knowledge making and production.

Our systematic plan of knowledge management includes the following aspects that need to be carefully addressed in the software development and transition processes:

- A systematic account of existence of knowledge.
  "What 'exists' is that which can be represented." When the knowledge about a domain is represented in a declarative language, the set of objects that can be represented, called the universe of discourse, must be clearly identified.
- An explicit and formal specification.
  This element concerns how to represent the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that forms the basis of knowledge representation. For example, we can describe the knowledge in terms of the ontology by defining a set of representational terms of a program. These definitions would associate the names of entities in the universe of discourse (e.g. classes, relations, functions or other objects), with human-readable text describing what the names mean and formal axioms that constrain the interpretation and well-formed use of these terms.
- A hierarchical structuring of knowledge.
  Suitable structures of knowledge representation include those constructs that organize relativities about data, information, and knowledge by subcategorizing them according to their essential (or at least relevant and/or cognitive) qualities in heterogeneous forms that accommodate both the static and dynamic features of knowledge management in spatial and temporal domains.
- A set of agents that share the knowledge.
  A good knowledge management scheme should be able to communicate about a domain of discourse without necessarily operating on a globally shared theory. For example, we say that an agent (a software entity) commits to the ontology of the specific domain if its observable actions are consistent with the definitions in the ontology. Nevertheless, the idea of ontological commitment is based on the clearly defined knowledge-level perspective.

III. The Three-tier Approach of Knowledge Management

III.1. A collaborative team organization

While software engineering processes exist as a multi-phased endeavor, each of the phases is a part of a larger picture that includes other elements residing within the whole process or in an organization that cannot exist in isolation. The most notable part of these elements, as pointed previously, is the people. To highlight the importance of human factors, experts in knowledge management working with the DoD have expanded the notion of knowledge management to one of "sense-making." Knowledge management and sense-making are critically important for future military operations. For example, getting the right information to the right people at the right time is central to Joint Vision 2020 [www.dtic.mil/jv2020], DoD's blueprint for future operations. It is also a key to the concept of network-centric warfare, which experts hope will bring future military sensors, decision makers, and shooters together on a shared network to improve shared awareness, speed of command, operations tempo, lethality, survivability, and self-synchronization. Again, the success of this mission is tightly linked to the knowledge management issues in C2 system operations.

Maintaining an appropriate level of knowledge in the software engineering support teams for C2 systems is a very important issue. It is, however, not an easy task for most organizations and it is particularly problematic for software development organizations, which are human and knowledge intensive.

III.2. A notion of continuous improvement process

We use the notions of continuous improvement and iterations as the main vehicle for planning, executing, evaluating, and improving software innovation processes. The processes include the development of sub-systems (organizations) dedicated for conceptual study of software innovation, for experimentation of innovative technology, and for evaluations of the soundness and maturity of the new technology. In a collaborative organization, the process for qualitative analysis of software innovations is as follows.

1. First, a set of theoretic concepts (suspected of containing important innovations) is identified by the engineering team according to their knowledge level. This is the *exploration* process of software innovation, and the knowledge involved appears as set of technology briefs or research reports.
2. Then, a set of key concepts is chosen that represents the topic area or issue that needs to be investigated. The techniques and theoretic foundations would be searched for the reasonability and applicability. This forms the *evaluation* process of software innovation. The stage includes some intellectual tasks of assessing the innovative technology ideas and looking for trends. Experiments would be conducted (including prototyping) to justify the applicability.
3. Finally, a new software package would be created to implement and test the innovation. This enters the *execution* process of the software innovation.

III.3. A three tier software engineering support structure

We adopt a three level structure of knowledge management for C2 software engineering support and innovation in corresponding to the three processes discussed in sections above. Thus, the three knowledge management levels for C2 software innovation are readily defined as:

1. The *Exploration Level*
2. The *Evaluation Level*
3. The *Execution Level*

It is apparent that tasks for the three levels of knowledge management are to be carried out in parallel to the three processes of a software innovation process. That is to say, the software engineering process and corresponding knowledge management structure is organized in a three-tier structure as shown in Figure 3. Naturally, all these levels should keep close interactions with the end-users (operators, systems engineers, and administrators) at every stage of the software innovation and technology transition processes.

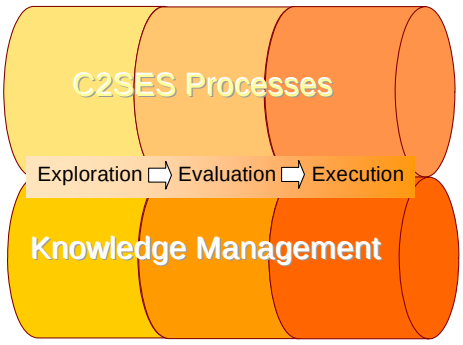

Figure 3 Three-tier structures of C2SES processes and knowledge management.

For example, tasks of knowledge management at these levels could be organized as the follows.

- The *Exploration* level is tasked to continually observe the technological trends, develop plans for C2 software innovation and improvement, and provide resources to C2 specific subject matter expertise.
- The *Evaluation* level is tasked to keep close eyes on the software product specifications for implementing and demonstrating the technical feasibility of innovations in a reconfigurable laboratory environment.
- The *Execution* level is tasked to carefully examine the software validity and certifications which include tasks of collecting feedbacks from field operations, and taking issues on trouble shooting, version tracking, and transition scheduling.

The notion of *continuous improvement and iterations* is the main vehicle for us to carry the planning, executing, evaluating, and improving cycles of the three-tier knowledge management scheme in C2 software innovation process. Built on the three-tier knowledge management scheme, the processes for qualitative introduction of software innovations in C2 software engineering then take the following steps.

1. First, identify a set of theoretic concepts (containing important innovations in principle) that is conducted by the innovation exploration team according to their knowledge level. An intellectual investigation of the ideas and their trends in development will be conducted. This would be the set of technology briefs or research reports.
2. Second, a set of key concepts is chosen that represents the topic area or issue that needs to be evaluated. The feasibility of the techniques and the theoretic foundations of the innovation will be studied for verifications of the reasonability and potential applicability accordingly.

3. Third, experiments will be conducted (including prototyping) to justify the applicability. The work will lead to a new software package created to implement and test the innovation idea.

We discuss the practical aspects of the three tier knowledge management scheme for C2 software innovation in next section.

IV. Practice of Knowledge Management in C2SES

IV.1. Organizational structure of knowledge management in C2SES

The Command and Control Software Engineering and Support (C2SES) project underway at United States Strategic Command (USSTRATCOM) requires a mix of technical expertise and qualifications for development and transition of software innovations. In these processes, organizational models are essential to knowledge management. As it is well said, "Organizational models set context." The characteristics of C2 systems call for a strict and conservative scheme of knowledge management in the entire engineering support process. By clearly understanding the linkages between process execution steps and identifying critical sources of external information helps establishing an overall taxonomy for the required knowledge.

The C2SES program is organized in line with the three-tier knowledge management structure that organizes the processes according to proper knowledge management levels. The multi-phased process aims at introducing software improvements through an incremental/evolutionary approach. The C2SES team uses the combined resources of academic research laboratories, software industrial technology centers, and military software integration laboratories to introduce innovative solutions to current problems, test interoperability among existing and proposed applications, and demonstrate promising technology solutions.

We emphasize the coupling of the software engineering processes and the knowledge management tasks, as well as the roles of people in making the connections. The C2SES managerial team understands the urgent needs (and has been successful) to attract and nurture knowledge workers and provide the environment for extraordinary thinking – for problem solving, executing complex engineering functions, and innovation (Table 1 [NGC05]). Knowledge relevant to each of these identities are abstracted, structured, and clustered in a suitable manner that facilitate its understanding, verification, validation, maintenance, management, testing, and interoperability.

Table 1. Roles, Responsibilities, and Authority in C2SES Software Development process

Role	Responsibilities
Systems Engineering (SE)	Develops and/or manages system-level requirements, requirements analysis, interface identification, management, and control, requirements traceability, Technical Performance Measures (TPMs), requirements allocation to hardware and software, system architecture, concept of operations, interface engineering, performance analysis, specialty engineering, integrated logistics support, models and simulations, and trade studies, and verification and validation.

Role	Responsibilities
Software Development (SW)	Establishes and maintains the architectural design, detailed design, implementation, and unit test of software components per design and implementation standards. Develops or supports the development of user documentation. Establishes and maintains Software Data Files (SDFs).
Test	Establishes and maintains test plans, test cases, and test procedures, integrates components into an operational system, conducts formal qualification tests, documents test results, and analyzes performance. Establishes and maintains Test Data Files (TDFs).
Quality Assurance (QA)	Ensures activities are conducted and products are produced in accordance with the contract, organizational policies, standards, and the defined process. Ensures quality products are delivered. QA retains an independent reporting chain to the division. The C2SES Mission Assurance Plan (MAP) defines the approach for QA activities.
Configuration Management (CM)	Manages configuration identification, configuration control, change management, configuration status accounting, and configuration audits of development artifacts. The C2SES Configuration Management Plan (CMP) defines the approach for CM activities.
Data Management (DM)	Manages control, receipt, delivery, distribution, and tracking of both deliverable and non-deliverable documents and records. The C2SES Data Management Plan (DMP) defines the approach for DM activities.
Process Group	Defines, oversees, and improves software development process assets, such as standards, procedures, templates, forms, tools, and the defined process tailored from the organizations standard process. Responsible for interfacing with the division Engineering Process Group (EPG).

The C2SES team closely monitors activities within the Knowledge Management domains and identify changes in the underlying hardware and software infrastructure, which may impact the operation and performance of USSTRATCOM C2 applications/tools under the purview of the C2SES program. The team is committed to proactively seeking out and recommending candidate C2 software engineering projects for research and demonstration to the user community. As infrastructure changes are identified, the team reviews any impact assessments and formulates a recommended test approach to include the amount of testing required to minimize risk to the program.

IV.2. Coupling the C2SES processes and knowledge management activities

IV.2.1 Knowledge management at exploration level

As the USSTRATCOM C2 mission evolves and matures, it is expected that the C2 applications and tools will also evolve to sustain, enhance, and optimize the C2 capabilities, as well as effect disciplined changes supporting mission requirements. The C2SES team recognizes that an Evolutionary Life-Cycle Model (Figure 4) fits well for these purposes.

The Evolutionary Life-Cycle Model is an iterative approach and thus has multiple cycles from requirements through system deployment. The number of iterations may vary based on schedule and operational needs or based on the depth and understanding of system requirements. As with any of the Life-Cycle Models, the Evolutionary Model can be tailored to meet the needs of the program, such as unique contractual requirements. For example, the Evolutionary Life-Cycle

Model can be tailored to reflect the Spiral Model through the introduction of risk reduction measures such as fast prototyping. This model is better suited for larger development activities or where risk is inherent in the development.

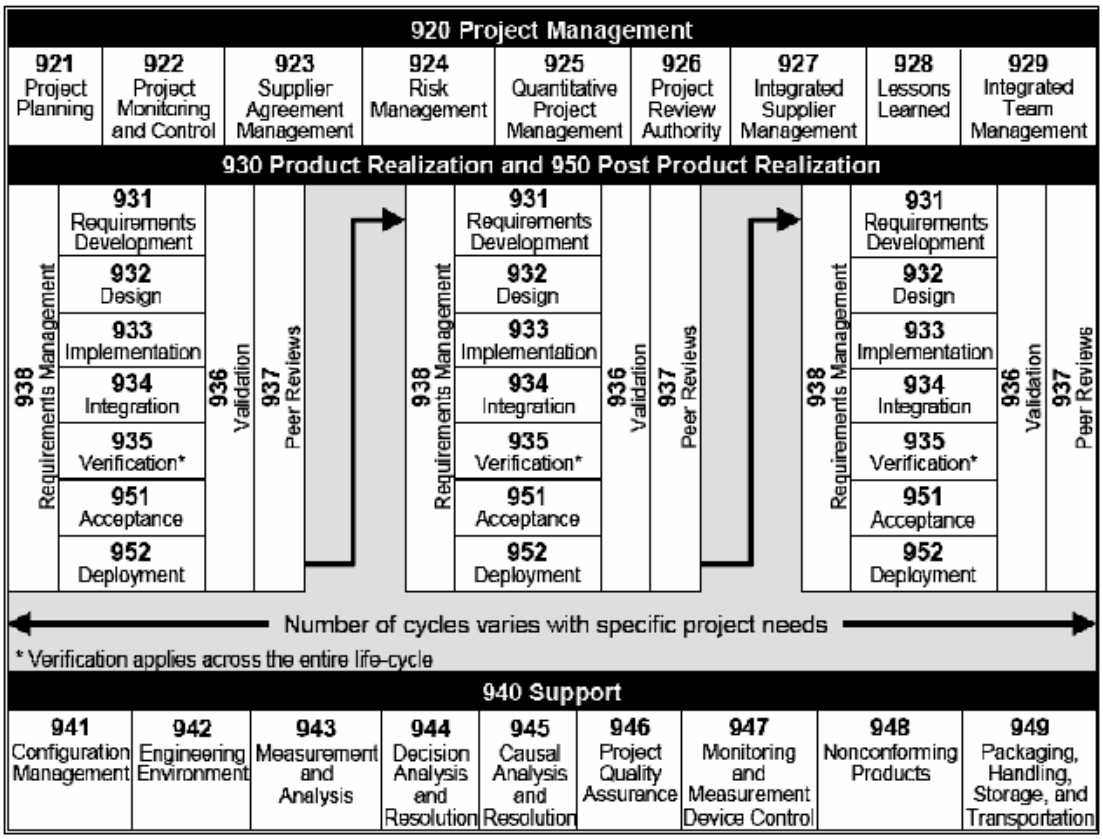

Figure 4. C2SES Evolutionary Life-Cycle Model

The three-tier knowledge management scheme allows knowledge to be shared more easily and consequently enables more collaborative software engineering support and executions. For example, research in agile methods and eXtreme Programming technique was conducted by scientists from Peter Kiewit Institute of Information Science, Engineering and Technology at the University of Nebraska. The emergence of intelligent, agent-based, adaptive software can greatly improve C2 capabilities at the operational level by providing decision support for both planning and execution. Intelligent agents that continuously monitor the events in the C2 environment can assist command center operations in providing information for staff planners to conduct threat analysis, terrain analysis, asset scheduling and tracking, route planning, logistics, fires coordination, communications, force protection, and coordination with allied troops and coalition forces.

The Evolutionary Life-Cycle Model calls for closely monitoring activities within the software engineering processes at the exploration level to identify changes in the underlying hardware and software infrastructure, which may impact the operation and performance of C2SES applications. As infrastructure changes are identified, the C2SES team reviews any impact assessments and formulate a recommended test approach to include the amount of testing minimizing risk to the program. During the planning stages of any significant software development effort, both

system and software engineering engage in a series of one or more concept exploration cycles to identify system and software level requirements. During concept exploration, risks are analyzed to determine whether a plan-driven or an agile development methodology will be implemented. When necessary, information used for this analysis is derived from prototyping, data collection, and other data analysis activities.

IV.2.2 Knowledge management at evaluation level

The C2SES process analyzes the system requirements allocated to software to identify the necessary software functional capabilities, performance requirements, control requirements, design constraints, and interface requirements. Additional requirements, i.e., derived requirements, are defined and documented as needed to complete the software requirements level of abstraction, e.g., to address topics such as user interfaces, safety, and security. Software requirements analysis develops a comprehensive set of specifications that serve as the basis for software development.

Knowledge maintained and managed in the evaluation level include

- Creating a Software Development Plan (SDP) to define the approach for developing, integrating, maintaining, and/or supporting software. The C2SES project uses the Northrop Grumman Mission Systems (NGMS) SDP Template. This plan establishes the common approach to be employed by all software development groups on the program, including subcontractors. This plan is used as the basis for managing software development activities during all life cycle phases of the program, and applies to newly developed, modified, reused, and acquired software. This plan is compliant with contract requirements, the NGMS Policy & Requirements Manual (PRM), and the NGMS Quality Management System (QMS).
- Updating the SDP due to changes in the contract, customer direction, program scope, requirements, available and estimated resources, organizational policies or processes, or when actual measurements vary significantly from the plan. Plans are reviewed for update at least annually and revised as required, to ensure consistency with the NGMS PRM and other program plans. More frequent reviews and updates are performed as necessary to ensure plans are current and useful. Updates are subject to the configuration and change management process defined in the C2SES Configuration Management Plan (CMP).
- Evaluating the risks of performing an abbreviated test, and provide appropriate software specification with a recommended test approach.
- Developing an architectural design to implement the approved software requirements and provide a sound foundation for software product design.

The software requirements specifications and interface requirements specifications are reviewed by all affected organizations, including system engineering, test, and quality assurance. The specifications are reviewed for understandability, feasibility, consistency, completeness, and testability. Reviews are also performed with customer and user representatives to validate their needs and expectations are clearly identified, understood, and prioritized. Knowledge entities in the evaluation tier are present in the C2SES software development resources. An excerption of the entries is shown in table 2.

Table 2. C2SES software development resource management (excerpt)

Tool	Application

PCTR	Policy Compliance and Tailoring for the Defined Process
e-Toolkit and PAL	Organizational Process Assets Repositories
Lessons Learned Database	Organizational Repository for Lessons Learned
Self Assessment Tool (SAT)	SEI CMMI Self Assessment
Risk Manager's Assistant	Risk Management
Software Development Plan Template (SM 921.3)	Software Development Plan Development (this plan)
Software Change Request Database	Change Management
Wiki Repository	Requirements Traceability
JDeveloper or Eclipse	Design Modeling, Debugger, Graphical User Interface (GUI) Builder
Java 2 Standard Edition Java Development Kit (J2SE JDK) 1.3	Compiler
Office of Cost Estimation and Risk Analysis (OCERA) Code Counter	Code Counter
McCabe	Cyclomatic Complexity

IV.2.3. Knowledge management at execution level

The software development process adheres to the guidance provided in the specification document, and is executed in conjunction with a variety of development activities in response to a range of events. The coupling of software development processes and the knowledge management activities at this level includes documentation of the software development and test results in the Software Production Documents and Test Reports, and place the data under configuration management control.

- The objective of *Software Detailed Design* activity is to complete the design of each software product or component identified in the approved software architecture. There are two design activities for each product: completing the detailed design of the interfaces and partitioning the product into software units. The software product design is reviewed by all affected organizations, typically in a series of incremental reviews. The design is reviewed for understandability, feasibility, consistency, completeness, correctness, and conformance to standards. Software engineering enters any identified problems in requirements, architecture, or design in the program's problem reporting tracking system
- The *Software Implementation and Unit Test* activity establishes and maintains the software code baselines for the software products in the approved software architecture and as defined by the product design. The principal activities are: refining the partitioning of the product design into units, developing the unit test plan, implementing the units in accordance with the program programming standards, and executing the unit test plan. The responsible engineer develops the code using the program-specified tools and standards. When the code compiles correctly, the engineer executes the unit test plan and verifies that the expected results are obtained.
- The *Software Integration* activity integrates software units into software builds to continue developmental software testing. Software builds are typically modes or major functional elements of the system. This activity results in a sequence of distinct builds, each with defined capability to accomplish development life-cycle objectives. The sequence and content of builds are chosen to mitigate risks and provide needed capability. The principal objective of the first build is typically to expose any problems or risk factors in the integration environment and procedures. Subsequent builds are chosen to expose any critical

risks related to the software design. The final build adds very little new functionality, but it is devoted to any updates related to requirements changes and software updates.

Knowledge management at the execution level also involves the following tasks.

- Document software build and installation instructions in each application Software Version Description (SVD). These instructions will provide sufficient detail so that system administrators will be able to build and deploy the C2 applications to the appropriate environment. An SVD will be provided with each software delivery.
- Prepare and maintain operator trouble-shooting guide. This guide will provide operators and help desk personnel with step-by-step instructions on resolving software operation and connectivity problems
- Implement and execute test processes designed to verify and validate application capabilities against defined requirements and to evaluate the impact of infrastructure changes on application performance.
- Work with the IPT (Integrated Product Teams) to prepare training materials to aid users, testers, and other USSTRATCOM organizations in the proper use of software, identify additional training materials necessary to support the installation, configuration, and monitoring the software applications, and update these materials with each major software release to reflect changes in application features and functionality.

The right metrics need to be selected to ascertain a selected knowledge management scheme accomplishes what the organization needs to accomplish. An important precursor to selecting right metrics for measurement of success is to examine whether the goal is reached. Success of knowledge management should be measured in combination of knowledge quantity (count of the number of axioms or assertions entered and validated per unit time) and knowledge quality (how accurate the error-free of the axioms and assertions). The process includes the adoption of commonly accepted software standards.

Depending upon the build content or customer direction, some technical reviews may be formal (i.e. a scheduled meeting), informal (i.e., via email or Wiki), or not required at all. Procedures for preparing and conducting Technical Reviews are documented in the SDSPM. The *Verification and validation* activities occur throughout the program life-cycle to ensure the delivered product meets requirements and customer and user needs. Products to be verified and the verification method to use are identified. Products to be validated and the validation method to use are identified in Table 3. The types and extent of validation to use ensures products meet customer and end user needs and requirements. The detailed schedule for verifications and validations of major product releases are documented.

Table **Error! No text of specified style in document.** C2SES Validation Method for Products

Product	Validation Method
Requirements	Technical reviews to ensure the customer and users review the requirements. Customer and user participation in technical working groups and simulations and models. Requirements are also validated against the customer/user requirements to ensure they correctly reflect the customer and users needs.

Product	Validation Method
Design	Technical reviews to ensure the customer and users review the design. Incremental demonstrations of the prototype to the customer and users to obtain feedback, including delivery of prototypes to the customer and users for more thorough review.
Implementation	Participation in code walkthroughs (not peer reviews, since the customer and user are not "peers").
Test	Peer reviews to ensure system-level tests are derived from customer needs (i.e., Concept of Operations (CONOPS)). Acceptance tests.
User Documents	Delivery of draft versions to the customers and users for initial feedback.

The following software documents carry the contents important to knowledge management at the development and execution level.

1. Technical Data Packages (TDPs) capture software development and test artifacts in a centralized location. TDPs are maintained electronically to the maximum extent possible via the C2SES Wiki Electronic Project Folders. When only hardcopy exists, such as signature pages, convert the material to electronic form via scanning, and provide the original hardcopy to CM for retention. TDPs provide customers, managers, and relevant stakeholders with visibility into software development and test status and are the principal working logs for software developers and testers. Each Configuration Item (CI) is required to have at least one TDP that may be further decomposed into component and/or unit level TDPs. For different builds or releases, create new TDPs or expand existing TDPs. The TDPs are reviewed periodically by APMs and WPMs to ensure they are maintained and kept up-to-date. The TDPs are randomly audited by QA to ensure compliance with policies, plans, and the defined process.

2. The Software Data File (SDF) is the software development organization's means of capturing software development related artifacts in a centralized repository. The SDF is established during the requirements phase and is maintained throughout the life of the program. The SDF virtually or physically contains the artifacts identified in Table 4.

Table 4. Software Data File (SDF) Contents

Section	Purpose
Cover Sheet	A cover sheet for each build or release of software that includes a high level description of the CI, component, or unit name, the responsible engineer, and due date and date complete columns for each section of the SDF. As each section of the SDF is completed, the cover sheet is updated and signed off to indicate development progress.
Requirements	List of software requirements applicable to the CI, component, or unit name. Include either copies of the applicable requirements or pointers to the applicable requirements. Requirements Traceability Matrix showing requirements applicable to the CI, component, or unit name.
Design	Design diagrams, design documents, interface descriptions, and/or threads and use cases applicable to the CI, component, or unit name.

Section	Purpose
Implementation	Location of the source code, make files, build scripts, and executables, preferably from a controlled CM library system, applicable to the CI, component, or unit name.
Test	Unit test plans, test cases, procedures, and test results indicating pass or fail for the CI, component, or unit name. Integration test and subsequent tests are stored in the TDF, not the SDF.
Problem Reports	List of problem reports submitted against the CI, component, or unit name which includes the status of each problem report. Copies of each PR may be included.
Reviews and Audits	Review material (presentations) and review results (agendas, minutes, and action items) from requirements, design, implementation, and test peer reviews, PDR and CDR technical reviews, management reviews, technical working groups, and non-conformances from audits.
Notes	Additional information deemed relevant to the CI, component, or unit name which is useful to developers, maintainers, or reviewers, such as trade studies, key design decisions, technical performance measurements, engineering notes, Interoffice Correspondences (IOCs), metrics reports, deviations and waivers, turnover memos, key e-mails, etc.

3. The Test Data File (TDF) is the test organization's means of capturing test related artifacts (excluding unit test) in a centralized repository. The TDF is established during the test phase and is maintained throughout the life of the program. The TDF virtually or physically contains the artifacts identified.

VI.3. Tools and mechanisms for knowledge management in C2SES

The whole process of knowledge management for software innovation discovery, assessment, testing, implementation, and installation would benefit greatly from tool support. For example, a Visual Query Interface can be helpful in identifying and organizing the experience packages to be used as the basis of the synthesis. Tools like Nvivo (from QSR International; more information at http://www.qsr.com.au/products/nvivo.html) can be used to automate the searching and coding, as well as facilitate searching for trends and packaging the results.

A trend in knowledge management today is to enable distributed teams of subject matter experts (SMEs) to enter and modify knowledge directly, easily, and without the need for specialized training in knowledge representation, acquisition, and manipulation. This was the goal of the DARPA's RKF (Rapid Knowledge Formation) program. The resulting tools of that program have been made available to provide specific answers to knowledge management questions and could be applied in many different software engineering and knowledge management processes and situations.

Most of the work carried out through the C2SES project is based on flows of information and knowledge, from one level of management to the next, from one worker to the other, and between the client and developer. Web-based collaborative software tool suite can be used to support C2SES knowledge management. Figure 5 illustrates the linkage by a construct of knowledge map to disclose data, information and knowledge assets of the various roles involved in the software engineering support and knowledge management processes.

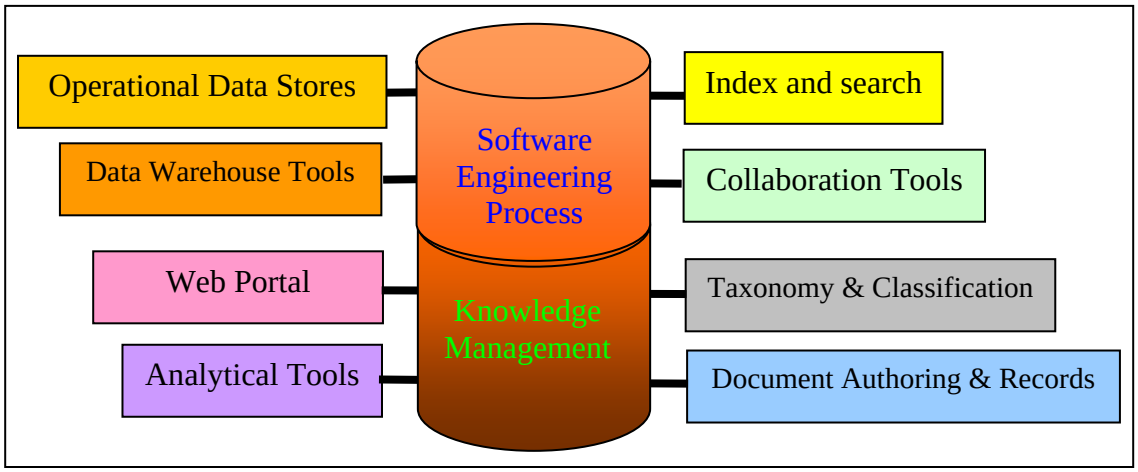

Figure 5. Web-based Knowledge Management Tools

It is seen from the above structure that much knowledge worker activities deal with keeping multiple instances of similar (and different) processes with multiple sources of active information in-flight at the same time. Effective systems need to be equally adapted at information and knowledge management levels as well as process execution and performance measurement phases. For example, software requirements analysis is performed using a storyboarding approach as a form of scenario-based requirements elicitation. Software requirements are developed using the use case analysis methodology. Software design is performed using the object-oriented design methodology

The C2SES software Wiki web-site functions as the repository for software requirements, traceability information, testability information, computer resources utilization budgets, and any other artifacts created during this activity. A web portal tailored to the C2SES management and the command and control personnel is utilized so that various software developers, engineers, and managers have access to the same information. A networked environment also promotes and facilitates the acquisition, sharing, and application of knowledge.

V. Conclusion

Knowledge management is a sound field with real benefits such as reduced training time for new employees, improved decision making and better operational efficiency. However, it is difficult to get it done right. One misconception about knowledge management is that all it is needed is about technology. What is the best method for capturing worker knowledge? Usually people begin a knowledge management project by focusing on the technology needs, whether they want a database or a portal. Technology can help capture some of the information but it is not the ultimate answer. The key is the people and the process. The one-size-fits-all mentality, coupled with the tendency to focus on technology rather than people and process, has obscured the real benefits that a good knowledge management scheme can bring. That is to say, technologies such as some software tools alone cannot solve the knowledge acquisition problem entirely. To have an effective knowledge management, it is necessary to adopt a systematic scheme of planning, stipulating, and distributing the knowledge management tasks and activities. This is a reason

that those who have successfully tackled knowledge management projects have taken a systematic approach. The approach treats the people and process as central pieces of the effort and addresses the relevant issues accordingly.

The area of information technology in C2 systems has seen several important developments in recent years. Among them are growing awareness and understanding of the importance of knowledge management to sound processes including software engineering and decision making. To highlight the importance of human factors, experts in knowledge management have expanded the notion of knowledge management to one of "sense-making." It is a key to the concept of network-centric warfare, which will bring future military sensors, decision makers, and shooters together on a shared network to improve shared situation awareness, speed of command, operation tempo, lethality, survivability, and self-synchronization. We hope the concept of layered knowledge management presented in this paper and the practice of the concept to C2 software innovation will offer an example to the effort of more efficient knowledge management in military applications.

We presented a systematic plan for introducing technology innovation to C2 software systems. We recognize that knowledge is broader, deeper, and richer than data or information. It is a mix of experience, value, contextual information, and expert insight that provides a framework for evaluating and incorporating new experiences and information. Knowledge comes from people. Our three-tier Knowledge Management scheme for C2 Software Innovation is a systematic plan of action for *bridging the technical and operational communities*. The objectives of the approach is to keep a balance among the factors of (1) smoothly and timely transforming legacy systems to updated, innovative, and modern systems, (2) maintaining continual, un-interrupted operation of the C2 software system, (3) minimizing software defect incidents, and (4) reducing or keeping C2 software maintenance and innovation at a controllable level of cost. By identifying needs for layered knowledge management, the C2SES team emphasizes the importance of using the *three tier infrastructure* to wring out potential new capabilities with thoroughly tested applications in a secured operational environment prior to fielding. We believe that it is an effective means, though not necessary the only means, for timely introducing technology innovations into mission-critic C2 software systems.

The knowledge management tenet of "getting the right information to the right people at the right time" and using it to make good decisions requires an understanding of human cognitive processes and capabilities as well as the technical means of retrieval, storage, discovery, and capture. By taking a proactive approach to knowledge management, the C2SES team is on a route to realize the improvements to the whole process in gaining and organizing knowledge toward innovations of C2 software systems.

References

[AL01] M. Alavi, and D. E. Leidner, "Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues," *MIS Quarterly, 25*(1), 107-136, 2001.

[AV02] Jan Achterbergh, and Dirk Vriens, "Managing viable knowledge," *Systems Research and Behavioral Science*, V19, i3, pp. 223, 2002.

[BCL01] Victor Basili, Patricia Costa, Mikael Lindvall, Manoel Mendonca, Carolyn Seaman, Roseanne Tesoriero, and Marvin Zelkowitz, "An Experience Management System for a Software Engineering Research Organization," *Fraunhofer Center for Experimental Software Engineering*,Maryland, 2001.

[BLC02] V. R. Basili, M. Lindvall, and P. Costa, "Implementing the Experience Factory Concepts as a set of Experience Bases," *Proceedings of 13th International Conference on Software Engineering & Knowledge Engineering*, pp.102-109, 2002.

[BW99] R. B. Brown, and M. J. Woodland, "Managing knowledge wisely: A case study in organizational behavior," *Journal of Applied Management Studies, 8*(2), 175-198, 1999.

[Chu71] C. W. Churchman, *The design of inquiring systems: Basic concepts of systems and organizations***,** Basic Books, Inc., New York, NY, 1971.

[CWP04] Marion G. Ceruti, Dwight R. Wilcox, and Brenda J. Powers, "Knowledge Management for Command and Control," proceedings of the 2004 Command and Control Research and Technology Symposium, San Diego, CA, June 2004.

[DB99] John Derrick, and Boiten, Eerke Boiten, "Testing Refinements of State-based Formal Specifications," *Software Testing, Verification, and Reliability*, 9(1):27-50, December 1999.

[DP98] T. Davenport and L. Prusak, *Working Knowledge,* Harvard Business School Press: Boston, MA, 1998.

[Du97] Soumitra Dutta, "Strategies for Implementing Knowledge-Based Systems," *IEEE Transactions on Engineering Management*, Vol. 44, No. 1, pp.79-90, Feb. 1997.

[KIN00] G. Krough, K. Ichijo and I. Nonaka, *Enabling Knowledge Creation*, New York: Oxford University Press, 2000.

[NT95] Ikujiro Nonaka and Hirotaka Takeuchi, *The Knowledge Creating Company*, New York: Oxford University Press, 1995.

[FV95] Michael A. Friedman, and Jeffrey M. Voas, *Software Assessment: Reliability, Safety, Testability*, John Wiley & Sons, Inc., 1995.

[Har03] S. Harvey, "Knowledge management: how do you do it?" *Training Journal*, July 2003.

[HAU00] Chin-Yu Huang, Sy-Yen Kuo, Michael R. Lyu, and Jung-Hua Lo, "Quantitative Software Reliability Modeling from Testing to Operation," *Proceedings of the Eleventh International Symposium on Software Reliability Engineering*, IEEE Computer Society Press, pp. 72-82, October 8-10, 2000.

[HP99] D. G. Hoopes, and S. Postrel, "Shared knowledge, "glitches" and product development performance," *Strategic Management Journal, 20*, 837-865, 1999.

[Ift03] Z. Iftikhar, "Developing an instrument for knowledge management project evaluation," *Electronic Journal of Knowledge Management, 1*(1), 55-62, 2003.

[JES00] Daniel R. Jeski, "Estimating the Failure Rate of Evolving Software Systems," *Proceedings of the Eleventh International Symposium on Software Reliability Engineering*, IEEE Computer Society Press, pp. 52-61, October 8-10, 2000.

[LYU96] Michael R. Lyu (Editor-in-Chief), *Handbook of Software Reliability Engineering*, Computer Society Press, McGraw-Hill, pp. 95-98, 1995.

[LOA00] Michelle Lee, A. Jefferson Offutt, and Roger T. Alexander, "Algorithmic Analysis of the Impacts of Changes to Object-oriented Software," *Proceedings of the Thirty Fourth International Conference on Technology of Object-Oriented Languages and Systems* (TOOLS USA '00), pp. 61-70, August 2000.

[LRJ01] M. Lindvall, I. Rus, R. Jammalamadaka, and R. Thakker, "Software Tools for Knowledge Management," *Proceedings of DACS State-of-the-Art Report*, 2001.

[Ma97] Y. Malhotra, "Knowledge Management in Inquiring Organizations," *Proceedings of the Americas Conference in Information Systems*, pp. 293-295, 1997.

[Ma01] Y. Malhotra, (Ed.), *Knowledge Management and Business Model Innovation*, Hershey: PA, Idea Group Publishing, 2001.

[Ma 04] Y. Malhotra, "Why Knowledge Management Systems Fail? Enablers and Constraints of Knowledge Management in Human Enterprises," In Michael E.D. Koenig & T. Kanti Srikantaiah (Eds.), *Knowledge Management Lessons Learned: What Works and What Doesn't*, Information Today Inc., pp. 87-112, 2004.

[McD99] R. McDermot, "Why information technology inspired but cannot deliver knowledge management," *Management Review, 5*(1), pp. 103-117, 1999.

[MM73] R. O. Mason, and I. I. Mitroff, "A Program for Research on Management Information Systems," *Management Science*, 19, 5, pp. 475-487, 1973.

[NGC05] Northrop Grumman Corporation, *Command and Control Software Engineering and Support (C2SES) Software Development Plan*, Prepared by David Richardson, 30 November 2005.

[OL98] Daniel E. O'Leary, "Knowledge-Management Systems Converting and Connecting," *IEEE Intelligent Systems*, pp.30-33, May/June 1998.

[PRR00] G. Probst, S. Raub, and K. Romhardt, *Managing Knowledge*. Chichester: Wiley, 2000.

[PS99] S. L. Pan, and H. Scarbrough, "Knowledge management in practice: an exploratory case study," *Technology Analysis and Strategic Management, 11*(3), pp. 359-374, 1999.

[Rec05] L. Russell Records, "The Fusion of Process and Knowledge Management," BPTrends, September 2005, www.bptrends.com (as of December 22, 2005).

[Ti00] A. Tiwana, *The Knowledge Management Toolkit: Orchestrating IT, Strategy, and Knowledge Platforms* (second ed.). USA: Prentice Hall, 2000.

[TV02] H. Touskas, and E. Vladimirou, "What is organizational knowledge," *Journal of Management Studies*, 3*8*(7), 973-993, 2001.

[Wa01] D. Wastell, "Barriers to effective knowledge management: Action research meets grounded theory [case study]," *Paper presented at the 9th European Conference of Information Systems*, Bled, Slovenia, June 27-29, 2001.

[Wi03] K. Wigg, Knowledge Management Foundations, ed. S. Press, Arlington, 2003.