View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by The University of Nebraska, Omaha

University of Nebraska at Omaha
DigitalCommons@UNO

Computer Science Faculty Publications

Department of Computer Science

9-2007

A three-tier knowledge management scheme for software engineering support and innovation

Richard Corbin

Christopher B. Dunbar

Qiuming Zhu
University of Nebraska at Omaha, qzhu@unomaha.edu

Follow this and additional works at: https://digitalcommons.unomaha.edu/compscifacpub

Part of the Computer Sciences Commons

Recommended Citation

Corbin, Richard; Dunbar, Christopher B.; and Zhu, Qiuming, "A three-tier knowledge management scheme for software engineering support and innovation" (2007). *Computer Science Faculty Publications*. 34.
https://digitalcommons.unomaha.edu/compscifacpub/34

This Article is brought to you for free and open access by the Department of Computer Science at DigitalCommons@UNO. It has been accepted for inclusion in Computer Science Faculty Publications by an authorized administrator of DigitalCommons@UNO. For more information, please contact unodigitalcommons@unomaha.edu.

A three-tier knowledge management scheme for software engineering support and innovation

Richard D. Corbin [a], Christopher B. Dunbar [b], Qiuming Zhu [c,*]

[a] *Northrop Grumman, Defense Mission Systems, Bellevue, NE 68005, United States*
[b] *FGM Inc., Omaha, NE 68142, United States*
[c] *College of Information Science and Technology, University of Nebraska at Omaha, Omaha, NE 68182, United States*

Abstract

To ensure smooth and successful transition of software innovations to enterprise 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 transition processes in levels from conceptual exploration to prototype development, experimentation, and product evaluation. The three-tier scheme is an integrated effort for bridging the development and operation communities, maintaining stability to the operational performance, and adapting swiftly to software technology innovations. The scheme combines experiences of academic researches and industrial practitioners to provide necessary technical expertise and qualifications for knowledge management in software engineering support (SES) processes.

Keywords: Knowledge management; Software engineering process; Software system support; Technology innovation; Human and system interaction

1. Introduction

This paper addresses the problems and issues of knowledge management (KM) in a software engineering support (SES) process. The problems are discussed in the context of how to maintain proper levels of knowledge in the process of developing and transiting technology innovations into enterprise software systems. The process typically includes

(1) insertion of new technology into existing software systems to enhance the system's operational capabilities,
(2) transformation of legacy systems to service-oriented, net-centric integrative systems, and
(3) installation of new systems to replace outdated systems, add novel capabilities to systems, and support closing the gaps in required capabilities of the systems.

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 (Russell Records, 2005)." However, what is "Knowledge" remains to be an issue in some people's mind, especially referring to specific domains and situations (Alavi and Leidner, 2001). 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

☆ An earlier version of this paper was presented at the 2006 Command and Control Research and Technology Symposium (CCRTS), San Diego, CA, June 20–22, 2006.
* Corresponding author. Tel.: +1 402 554 3685; fax: +1 402 554 3400.
E-mail addresses: Corbinr@stratcom.mil (R.D. Corbin), dunbar@fgm.com (C.B. Dunbar), qzhu@mail.unomaha.edu (Q. Zhu).

0164-1212/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2007.01.013

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 (Wigg, 1993). It is true, though, in certain circumstances where a piece of knowledge to one group of people could just be a piece of information to other group of people. But this argument cannot be generalized.

Information systems researchers (Churchman, 1971; Mason and Mitroff, 1973; Malhotra, 1997) have discussed various knowledge model, particularly for environments characterized by complexity, uncertainty, and radical changes. Churchman (1971), 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 (Nonaka and Takeuchi, 1995) 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 (Davenport and Prusak, 1998) 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 Fig. 1 for a denotation of data–information–knowledge hierarchy) (Brown and Woodland, 1999). 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.
3. Knowledge is most likely distributed among people and locations, often tailored to specific domain and practices, usually valid only at certain circumstances and within certain period of time, and subject to individual's view for its value and usage.

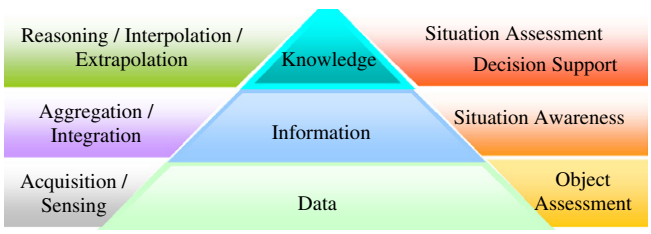

Fig. 1. Data–information–knowledge hierarchy.

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 intelligence traits. 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 mind and reasoning must be conveyed. Knowledge builds upon the foundation established by information and is, by way of contrast, people-oriented and mental work 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.

It has been said that the emergence of new technologies are making software more and more powerful, but at the same time problems and issues with new technologies are "every project manager's worst nightmare" (Brössler, 1999). The constant change and stream of new technologies makes it "very difficult to keep the organization ahead in the competition" (Tiwana, 2000). While time should be spent on actively searching for knowledge both within the organization and outside, software organizations seem to have even less time than others to do that because of the fast pace of the business (Basili et al., 2001). To render successful transition and capability improvement in software innovation, it is critical to maintain proper levels of knowledge within and around the software engineering processes both inside and outside an organization. These include the knowledge about the software systems, the operating environment, the development processes, and the technologies applied 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 systems and product transitions, avoid unmature 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 software engineering support (SES) 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 discovered 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, ensure minimal impact to the operational performance, and promote swift adaptation of new technology to the system operations.

In the remainder of this paper, we discuss the major issues of knowledge management in software engineering support, innovation, and performance improvement processes in Section 2. Section 3 presents the key concepts of a three-tier knowledge management scheme for a typical software innovation and transition process. Section 4 presents an exemplar software engineering support project applying the three-tier scheme at an enterprise system innovation level. We conclude the paper with a summary of the scheme and approach in Section 5.

2. Knowledge management in software engineering support and innovation

2.1. Knowledge management issues in software innovation

As we all understand that software engineering is a complex, dynamic, and multi-phased knowledge intensive process (Jeski, 2000; Basili et al., 2002). Knowledge in software engineering process is distributed at different levels – from individual, to project, to organization, to industry. No single knowledge repository would be sufficient to cover the knowledge necessary for all phases, stages, and the entire life cycle of the software engineering process.

The tasks of maintaining appropriate levels of knowledge in the software engineering process consist of many facets. Most of these facets 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 knowledge (e.g., in the form of experiences) no matter how well the knowledge has been documented.
(2) Latency of knowledge acquisition – To gain knowledge requires time. For example, when new employees are hired they need to get up to speed by acquiring technical and subject matter knowledge.
(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 (Hoopes and Postrel, 1999; Huang et al., 2000). 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 in their entirety (McDermot, 1999). 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, effective and dynamic component of the software engineering processes. That is, consider knowledge management as an inseparable dimension in the whole software development, transition, and maintenance processes. This leads to a scheme known as taking knowledge representation as "information in action" (Malhotra, 1997).

2.2. 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 (COP) 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 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), what knowledge the person has, and how extensive the knowledge was applied. Knowledge management schemes need to work adaptively in the whole range of the tasks and the whole processes and 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 software innovation process, and take a look at what processes it consists of. Namely, at a properly abstracted level (Fig. 2), these processes are roughly divided according to the activities involving

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.

2.3. 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. Often, moderating and intervening variables play a significant role in KMS upon correlating them with business performance outcomes. The dynamic, evolutionary nature of knowledge management calls for a system architecture that anticipates change and that fosters the systematic injection of upgraded systems, sub-systems and components. For example, one scheme of knowledge management in 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 software engineering and technology innovation "does not simply exist – people create it (Malhotra, 2001)." A systematic plan of knowledge management can help people do better through its impact on knowledge acquisition and maintenance if it necessarily addresses the following issues:

- *A systematic account of existence of knowledge* – "What 'exists' is that which can be represented (Malhotra, 2001)." 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 entities that are assumed to exist in area of interest and the relationships that forms the basis of knowledge representation. For example, we can describe knowledge in terms of ontology by defining a set of representational terms of processes. These definitions would associate the names of entities in the universe of discourse, with formal axioms for their interpretations.
- *A hierarchical structure of knowledge* – Suitable structures of knowledge representation include those constructs that organize relativities about data, information, and knowledge by sub-categorizing them according to their

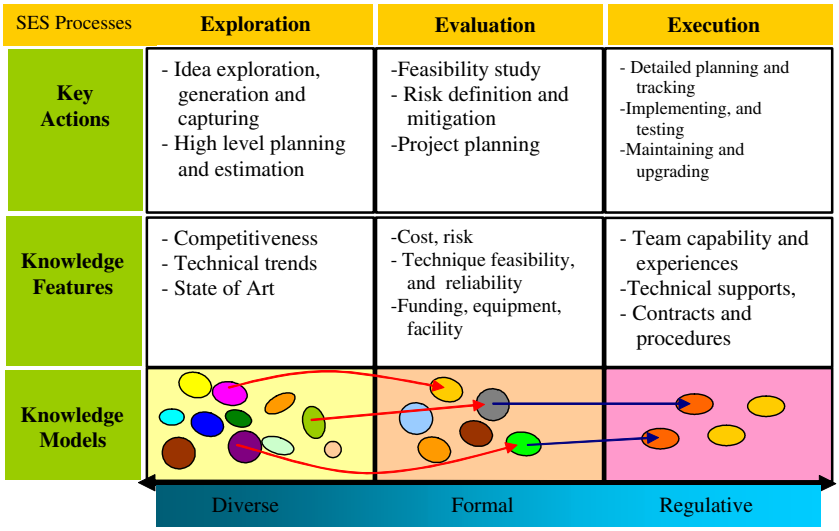

Fig. 2. Knowledge management in a multi-phased software innovation process.

essential 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, an agent (a software entity) is committed to ontology of a specific domain if its observable actions are consistent with the definitions in the ontology. The idea of ontological commitment is based on the clearly defined knowledge-level perspective.

3. Three-tier scheme of knowledge management for SES and innovation

3.1. A notion of continuous improvement process

We take the notion of *continuous improvement and development iterations* as the main vehicle to carry out the planning, executing, evaluating, and improving cycles of an software innovation process. Built on a collaborative organizational structure, the process for qualitative introduction of software innovations takes the following steps:

1. First, identify a set of theoretic concepts (containing important innovations in principle) that is conducted by an innovation exploration team according to their knowledge level. An intellectual investigation of the ideas and their trends in development is conducted. The outcome will be a set of technology briefs or research reports.
2. Second, a set of key concepts is chosen that represents the topic area or issues that need to be evaluated. The feasibility of the techniques and the theoretic foundations of the innovation is studied for verifications of the feasibility and potential applicability accordingly.
3. Third, experiments are conducted (including prototyping) to justify the applicability. The work will lead to a new software package created to implement and test the innovative idea and technological feasibility.

Tasks involved in these steps can be further decomposed into functional blocks as shown in Fig. 3.

It is noted that knowledge required for these three steps are quite distinction in terms of the related domain of expertise. It is hard to have a common knowledge repository (which may refer to human experts) to possess the knowledge necessary for all these areas, for example, considering the knowledge gaps among the developers, supporters, and operators of an enterprise software system.

3.2. A three-tier software engineering support structure

It is apparent that there is a requirement for a mixture of technical expertise and qualifications for development and transition of software innovations. Organizational models are essential to knowledge management in these processes, as it is well said, "Organizational models set context (Malhotra, 2001)." The characteristics of software engineering support and innovation process calls for a strict and conservative scheme of knowledge management at different levels of the technological domain and practice. By clearly understanding the linkages between the process steps and identifying critical knowledge sources helps to establish an overall taxonomy for the required knowledge management domain and levels.

Based on the above notion, we see a need for a three level structure of knowledge management for software engineering support and innovation in corresponding to the three processes above. The three knowledge management levels for software innovation thus can be 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 parallel structure as shown in Fig. 4. Naturally, all these levels should keep close interactions with the

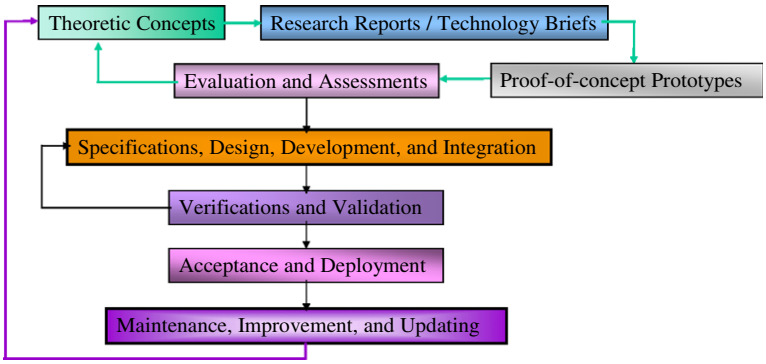

Fig. 3. Diagram of software system innovation processes.

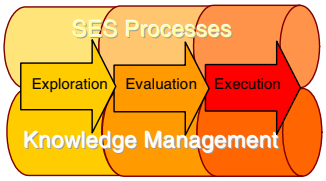

Fig. 4. A parallel process of the three-tier knowledge management and SES processes.

end-users (operators, systems engineers, and administrators) at every stage of the software innovation and technology transition processes.

Tasks of knowledge management at these levels can be allocated as follows:

- The *Exploration* level is tasked to continually observe the technological trends, develop plans for software innovation and improvement, and provide subject matter expertise.
- The *Evaluation* level is tasked to provide a feasibility report and software product specifications for implementing and demonstrating the technical innovations.
- The *Execution* level is tasked to develop the software prototype, test and collect feedbacks from field operation, and conduct trouble shooting, version tracking, and transition scheduling.

The three-tier knowledge management scheme allows knowledge to be shared easily. It consequently enables more collaborative software engineering support and executions. The emergence of intelligent, agent-based, adaptive software can greatly improve capabilities at the operational level by providing decision support for both planning and execution. Intelligent agents that continuously monitor the events in the environment can assist SES operations in providing information for planners to conduct market analysis, asset scheduling, logistics, communications, and coordination with different departments within the organization. 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 between software innovation and operational stability. Knowledge relevant to each of these steps are abstracted, structured, and clustered in a suitable manner that facilitate its understanding, verification, validation, maintenance, management, testing, and interoperability. Fig. 5 shows the relative knowledge management effort needed at each level of the software development and innovation process.

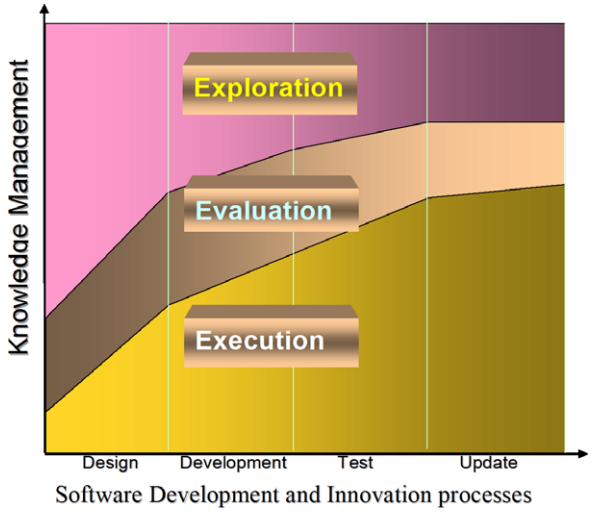

Fig. 5. Knowledge management efforts needed at different stages of software development and innovation.

We discuss the practical aspects of the three-tier knowledge management scheme for software engineering support and innovation in next section.

4. Practice of three-tier knowledge management in SES and innovation

4.1. Organizational structure of knowledge management in SES

The Software Engineering and Support (SES) project underway in our distributed cooperative setting is organized in line with the three-tier knowledge management structure. The SES managerial team establishes the specifications to acquire and nurture knowledge necessary for executing complex engineering functions and innovation (Table 1).

The SES innovation 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 the organization's applications/tools under the purview of the SES program. The team is committed to proactively seeking out and recommending candidate 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 minimize risk to the program.

4.2. Coupling the SES processes and knowledge management activities

4.2.1. Knowledge management at exploration level

As an enterprise's business evolves and matures, it is expected that the software applications and tools will also evolve to sustain, enhance, and optimize the capabilities, as well as effect disciplined changes supporting business requirements. The SES team recognizes that an Evolutionary Life-Cycle Model (Fig. 6) 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

Table 1
Roles, responsibilities, and authority in SES process

Role	Knowledge management 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
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. A SES 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. A SES 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. A SES 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)

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.

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 SES applications. As infrastructure changes are identified, the SES team reviews any impact assessments and formulates 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.

4.2.2. Knowledge management at evaluation level

The SES 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. This plan establishes the common approach to be employed by all software development groups in the program, including sub-contractors. 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 Policy & Requirements Manual (PRM), and the 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 PRM and other program plans. More frequent reviews and updates are performed as necessary to

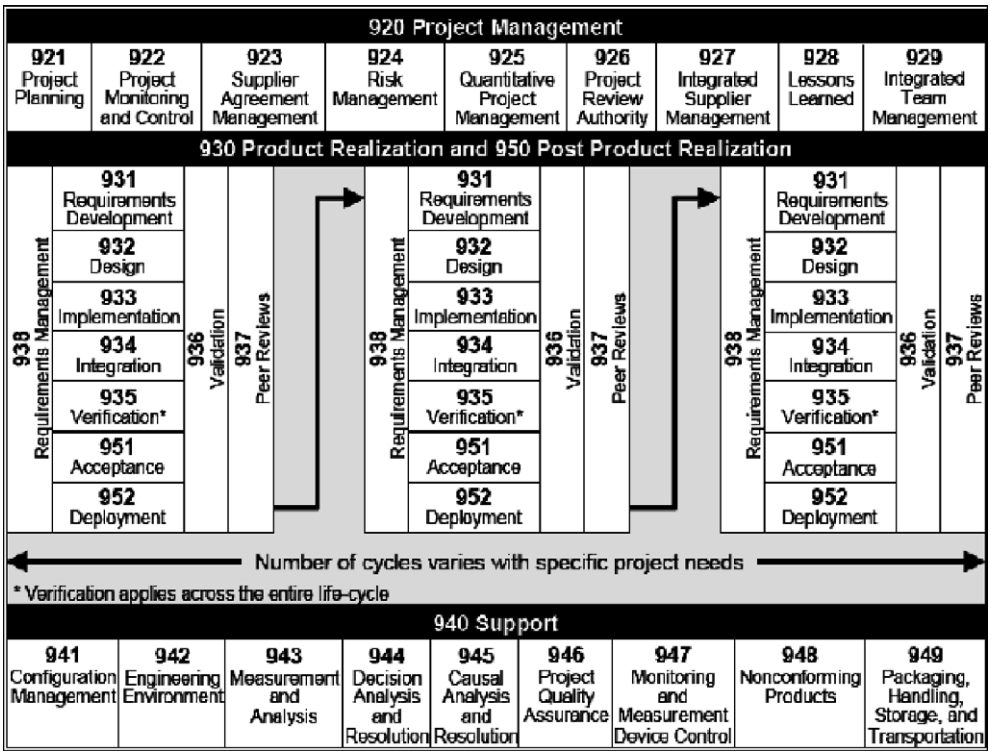

Fig. 6. An adopted SES and innovation evolutionary life-cycle model.

ensure plans are current and useful. Updates are subject to the configuration and change management process defined in the SES 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 make sure that expectations are clearly identified, understood, and prioritized. Knowledge entities in the evaluation level are present in the SES software development resources. An excerption of the entries is shown in Table 2.

4.2.3. Knowledge management at execution level

The software development process should adhere 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 the *execution* 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 testing engineers enter any identified problems in requirements, architecture, or design in the program's problem reporting and tracking system.
- The *Software Implementation and Unit Test* activity establishes and maintains the software code baselines in the approved software architecture and as defined by the product design. The principal activities are: refining the partitioning the product design into units, developing the unit test plan, implementing the units in accordance with the programming standards, and executing the unit test plan. The responsible engineers develop the code using the program-specified tools and standards. When the code compiles correctly, the engineers execute the unit test plan and verifies that the expected results are obtained.

Table 2
SES 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
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

- The *Software Integration* activity integrates software units into 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 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 helpdesk personnel with step-by-step instructions on resolving software operation and connectivity problems.
- Work with the IPT (Integrated Product Teams) to prepare training materials to aid users, testers, and other organizational units 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 for measuring the success need to be carefully selected to ascertain a 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 SPM. 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 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 at the same time.

The following software documents carry the contents important to knowledge management at both the evaluation and execution levels:

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 SES Electronic Project Folders. 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 project managers 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.

Table 3
SES knowledge management for product validation (excerpt)

Product	Knowledge management for validation
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
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

2. The Software Data File (SDF) is the software development organization's means of capturing software development related knowledge 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.
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.

4.3. Tools and mechanisms for three-tier knowledge management in SES

The whole process of three-tier knowledge management for software innovation can be benefit greatly from automatic tool support (Lindvall et al., 2001). A trend in knowledge management tools 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.

Most of the tools useful for carrying out the knowledge management tasks in SES can be categorized according to the 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 suites are the most common products used to support SES knowledge management. Fig. 7 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.

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 management mechanisms need to be equally adapted at different information and knowledge management levels as well as process execution and performance measurement phases. For example, a software requirement is developed using the use case analysis methodology, a software

Table 4
Knowledge management in software data file (SDF) contents (excerpt)

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
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.

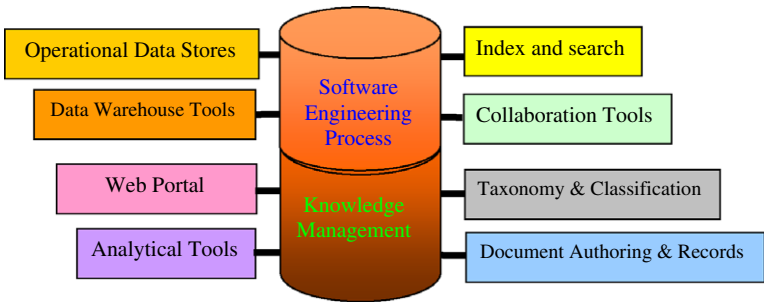

Fig. 7. Web-based knowledge management tools (conceptual).

requirement analysis is performed using a storyboarding approach as a form of scenario-based requirements elicitation, and a software design is performed using the object-oriented design methodology. Each of these mechanisms is associated with a rich set of knowledge management tools and facilities.

5. 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 (Malhotra, 2004). 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 three-tier approach discussed in this paper treats the people and process as central pieces of the effort and addresses the relevant issues accordingly.

We presented a systematic plan for introducing technology innovation to software systems in this paper. 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. Our three-tier Knowledge Management scheme for 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 software system, (3) minimizing software defect incidents, and (4) reducing cost or keeping software maintenance and innovation at a controllable level of cost. By identifying needs for layered knowledge management, the SES team emphasizes the importance of using the *three-tier scheme* 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 enterprise software systems.

References

Alavi, M., Leidner, D.E., 2001. Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. MIS Quarterly 25 (1), 107–136.

Basili, V., Costa, P., Lindvall, M., 2001. An experience management system for a software engineering research organization. Fraunhofer Center for Experimental Software Engineering, Maryland.

Basili, V.R., Lindvall, M., Costa, P., 2002. Implementing the experience factory concepts as a set of experience bases. In: Proceedings of 13th International Conference on Software Engineering & Knowledge Engineering, pp. 102–109.

Brössler, P., 1999. Knowledge management at a software engineering company – an experience report. Workshop on Learning Software Organizations, LSO'99, pp. 163–170.

Brown, R.B., Woodland, M.J., 1999. Managing knowledge wisely: A case study in organizational behavior. Journal of Applied Management Studies 8 (2), 175–198.

Churchman, C.W., 1971. The Design of Inquiring Systems: Basic Concepts of Systems and Organizations. Basic Books, Inc., New York, NY.

Davenport, T., Prusak, L., 1998. Working Knowledge. Harvard Business School Press, Boston, MA.

Hoopes, D.G., Postrel, S., 1999. Shared knowledge, glitches and product development performance. Strategic Management Journal 20, 837–865.

Huang, C., Kuo, S., Lyu, M.R., Lo, J., 2000. Quantitative software reliability modeling from testing to operation. Proceedings of the Eleventh International Symposium on Software Reliability Engineering, October 8–10. IEEE Computer Society Press, pp. 72–82.

Jeski, D.R., 2000. Estimating the failure rate of evolving software systems. Proceedings of the Eleventh International Symposium on Software Reliability Engineering, October 8–10. IEEE Computer Society Press, pp. 52–61.

Lindvall, M., Rus, I., Jammalamadaka, R., Thakker, R., 2001. Software tools for knowledge management. In: Proceedings of DACS State-of-the-Art Report.

Malhotra, Y., 1997. Knowledge management in inquiring organizations. In: Proceedings of the Americas Conference in Information Systems, pp. 293–295.

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

Malhotra, Y., 2004. Why knowledge management systems fail? Enablers and constraints of knowledge management in human enterprises. In: Koenig, Michael E.D., Kanti Srikantaiah, T. (Eds.), Knowledge Management Lessons Learned: What Works and What Doesn't. Information Today Inc., pp. 87–112.

Mason, R.O., Mitroff, I.I., 1973. A program for research on management information systems. Management Science 19 (5), 475–487.

McDermot, R., 1999. Why information technology inspired but cannot deliver knowledge management. Management Review 5 (1), 103–117.

Nonaka, I., Takeuchi, H., 1995. The Knowledge Creating Company. Oxford University Press, New York.

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

Tiwana, A., 2000. The Knowledge Management Toolkit: Orchestrating IT, Strategy, and Knowledge Platforms, second ed. Prentice Hall, USA.

Wigg, K., 1993. Knowledge Management Foundations: Thinking About Thinking – How People and Organizations Create, Represent, and Use Knowledge. Schema Press, Arlington.