

Most end users of software-intensive systems think they are buying the *system* and its capabilities, and not the *software*. It has become harder to separate systems engineering from software engineering because both must usually be involved in the development of the system. Software is a means for providing capabilities. Software accounts for 80 percent of weapon system functionality. The same holds true for almost all our electronic gadgets from radios to cell phones to microwave ovens.

Problems with the cost and schedule for delivery of systems are often blamed on software. But software

What Is “Software Engineering,” and What Isn't?

problems are now system problems and vice versa. For this reason, this issue of *Collaborations* presents a small sample of innovations in software engineering that are being used at MITRE and elsewhere. Software engineering is loosely defined here as technical as well as management activities, process as well as product approaches in acquisition, as well as development, delivery, and sustainment. Recent innovations in software engineering include use of commercial-off-the-shelf products and open source software; service-oriented architecture; model-driven architecture; enterprise architecture and integration; patterns; and agile, eXtreme, and spiral acquisition and development.

Model-Driven Architecture

The Object Management Group's Model-Driven Architecture (MDA) is a new spin on tools for developing software. Incorporating the principles of open architecture, model-based design, and agile development, MDA implements a system from a model of the system.

The system's structure and behavior are defined in a Platform Independent Model (PIM) in terms of the constructs of Unified Modeling Language (UML) with action semantics (xUML). The PIM is described in a higher level of abstraction than traditional programming languages and is independent of any particular middleware or programming language. Because the PIM is not tied to a specific computational infrastructure, the model

retains its value even when middleware and language evolve.

After the PIM is sufficiently specified, it is verified in a simulator. A model compiler completes the transformation of the PIM to the selected language and middleware, which is compiled for the target platform.

Though many consider MDA a method for rapidly producing software, its biggest return on investment may be productivity savings in software adaptation, reuse, and maintenance.

MITRE has been involved in helping to plan for and applying MDA on a military system.



For more information on MDA, contact John Castleberry, jcastleberry@mitre.org,

703-983-6435 or James Watkins, jcwatkins@mitre.org, 703-983-7868.



Osprey: Understanding Object-Oriented Software



Developing tools to understand object-oriented software is an important research area. Software analysis helps to answer questions about maintenance, reuse, code upgrades, and component integrations, as well as to identify hidden and explicit requirements.

Of the many techniques for understanding object-oriented software structures, design patterns are an obvious choice. The design pattern community has developed best practice descriptions for creating object-oriented

software. These descriptions are agreed-upon knowledge that bridges software constructs, their rationale, and their consequences for code quality.

MITRE has developed a prototype software system called the Object-oriented Software Pattern REcoverY, or Osprey. Osprey uses 104 recognizers (declarative descriptions of the constraints that must be met if we are to conclude that a pattern is present in the code) to identify occurrences of design patterns in the software. When given a program to analyze, Osprey matches the relationships among that program's structures and the prebuilt design pattern templates.

One design pattern, the Adapter, is used to adapt a new service to meet the interface of a preexisting set of services. For example, someone might have a set of services that works with x-y coordinates, but the program needs to take advantage of a service that produces results in distance-and-bearing coordinates.

With Osprey, analysts can look at programs in a more informed way. They can make statements such as "The purpose of class A is to adapt the results produced by class B in order to match the interface defined by class C." They can also make statements about code quality. We identified source code qualities (e.g., reliability, evolvability, and performance) affected by the use of design patterns and added this information to Osprey.

Osprey generates HTML documentation based on all of the analysis it does on the source code. The generated documentation consists of:

- Code-level reports (classes, methods)
- Recovered design patterns
- Software quality and design rationale inferred from the use of patterns
- Mappings from high-level expectations down to the source code that meets those expectations

Using Osprey Output

Analysts can start with program structure, with design pattern instances, with quality issues, with vulnerability concerns, or with design concepts. From any of these starting points, analysts can navigate to design patterns and to color-coded source code files.

- To fully understanding how the program is structured, start with design patterns.
- To assess the software quality for source selection, reusability, or interoperability, start with quality reports or perhaps reports that show use of operating system services.
- To assess the potential vulnerabilities of the code, start with any of a collection of vulnerability reports.



To learn more about Osprey technology, see "Relating Expectations to Automatically Recovered Design Patterns" by A. Asencio, S. Cardman, D. Harris, and E. Laderman, published in the *Proceedings of the Working Conference on Reverse Engineering* (Richmond, Va., 2002) and available from the authors, who can be reached at osprey@mitre.org.

Implementing Web Services on Mission Critical Systems: State of the Practice

Industry and government have embraced Service-Oriented Architectures and Web services as the answer to providing greater integration, interoperability, and adaptability across an enterprise of systems that may have been independently developed and that may change over time. Through time-honored system engineering principles such as encapsulation and integration via interfaces, Service-Oriented Architectures can alter software engineering in significant ways. The system and software architecture must be structured to make use of Web services, which, in turn, increases the reuse of software components. Services must have well-defined interfaces, follow commercial standards such as Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL), and be widely accessible via common protocols such as hypertext transfer protocol (HTTP). Access to services must be independent of how they themselves were internally implemented. Web services impact the business model for developing and reusing software. Rather than reusing software by moving it from system to system (e.g., by porting), services enable the reuse of software by making it accessible over a network. Industry organizations such as World Wide Web Consortium (W3C) and Organization for the Advancement of Structured Information Standards (OASIS) have taken on the job of standardizing Web services. There must be a market for services that offers incentives for their development. Lifecycle responsibility for

the management of service quality, performance and security must be assigned.

Although the concept is appealing, there are business and technology challenges. Here are some early findings from implementing Web services:

- What is the right level of service granularity? How much should be exposed in the Application Programming Interface (API) and in the data schema to retain simplicity of use and independence?
- How is meaningful information communicated through services? How are services tied to efforts to establish common semantics and vocabularies across systems?
- What is the best way to register services in directories that make them visible and accessible to a wide enterprise community?
- Web services must work over a heterogeneous set of communications networks, platforms, data, and software. How can compatibility be achieved without consuming valuable processing time and bandwidth?
- Who is responsible for assuring the services are reliable and secure over the life of the enterprise? Who will maintain them when the enterprise changes?

Programs are just beginning to build Web services using the simplest Web service standards. Web services are beginning to be catalogued and registered using technologies such as Universal Description, Discovery and Integration (UDDI). Some pro-

grams are reverse engineering Web services from legacy components by exposing their application interfaces and the database schemas. Though this approach may result in quicker production of new services, there is a danger that reverse engineered services may be tightly coupled to old programming interfaces and result in brittle implementations. Service design and granularity are still major issues.

Service-Oriented Architectures can alter software engineering in significant ways.



For more information about Service-Oriented Architectures and Web services, contact

Raymond A. Modeen, modeen@mitre.org,
781-266-9627.

The Government Spotlight on Software

A number of edicts have been issued by the government over the last several years to mandate improvements in software acquisition. The initial requirement was Section 804 of the Bob Stump National Defense Authorization Act for Fiscal Year 2003. It requires the establishment of software acquisition process improvement programs throughout the Military Departments and the Defense Agencies that manage Major Defense Acquisition Programs with a substantial software component. The Services have responded with their plans. In this issue we present summaries of the Army and Air Force responses to Section 804.

In March 2004, the General Accounting Office issued a report (GAO-04-393) entitled “Stronger Management Practices Are Needed to Improve DoD’s Software-Intensive Weapon Acquisitions.” The study compares the Department of Defense’s (DoD’s) software process improvement programs with the practices of leading companies and makes recommendations related to the management of software requirements, deliverables resulting from a disciplined process, collection of metrics monthly and prior to major milestones on cost, schedule, size, requirements, tests, defects, and quality of software. The study also recommends that the DoD’s acquisition policy, software acquisition improvement plans, and development contracts should enforce and incentivize these recommendations.

“Software is the
It is fundamental
and national security

Air Force Policy for Software Acquisition Process Improvement

In response to Section 804, Software Process Acquisition Improvement, the Air Force formed the SAF/AQ Air Force Software-Intensive Systems Strategic Improvement Working Group (AFSSIP WG). This group drafted and received approval for its AF 804 Strategic and Implementation Plans.

One outcome of their work is Air Force policy memo 04A-003, “Revitalizing the Software Aspects of Systems Engineering,” which was issued on September 20, 2004, by Peter Teets, the undersecretary of the Air Force, and Dr. Marvin Sambur, the assistant secretary for acquisition. The memo specified the following software focus areas throughout the life cycle of the acquisition of a software-intensive system:

1. High-confidence estimates of software development and integration effort, cost, and schedule (80–90% confidence)
2. Realistic program baselines
3. Risk management specific to computer systems and software
4. Capable developer of software
5. Developer processes that are consistently applied to effective software development
6. Program Office processes that are effective for software acquisition, are adequately staffed, and consistently support the developer in the disciplined application of established development processes
7. Earned Value Management applied to software
8. A core set of software metrics to manage software development
9. Lifecycle software support needs addressed during design and development phases
10. Lessons learned and data transferred to the Acquisition Center of Excellence

In support of this memo, the AFSSIP WG is developing a Software Acquisition Handbook that contains more detail on the memo and will be ready for distribution in early 2005.



An enclosure to the policy memo contains core software management metrics. For a complete copy of the memo, send email to Linda Scannell, scannell@mitre.org, 781-271-3358.

new physical infrastructure of the information age.
critical to economic success, scientific and technical research,
security.” — Report of the President’s Information Technology Advisory Committee, 24 February 1999

Army Strategic Software Improvement Program

In concert with congressional direction in Section 804, the Department of Defense Services have been making plans to improve their software acquisition practices. Realizing that software is critical to the capabilities of all defense systems and is growing in size and complexity as the Army moves to the Future Force, Claude M. Bolton, Jr., Assistant Secretary of the Army (Acquisition, Logistics and Technology) (ASA (ALT)) initiated

The PEO IEW&S continues to emphasize its goal to improve its software acquisition performance.

ated the Army Strategic Software Improvement Program (ASSIP). ASSIP is a long-term (FY03-FY09) commitment to dramatically improve the acquisition of software-intensive systems—people, programs, production/sustainment, and to provide continuous improvement. ASSIP is a partnership for improvement among ASA (ALT), Program Executive Officers (PEOs)/Program Managers, Software Engineering Centers, and the Software Engineering Institute, Carnegie Mellon University. Within James T. Wessel’s role on PEO Intelligence, Electronic Warfare, and Sensors (IEW&S) Staff, MITRE

provides PEO IEW&S representation to the ASSIP.

ASSIP chair, Jim Linnihan has championed the need for focused integration of systems and software engineering. Key program management topics related to software engineering include:

- Clearly defining requirements and expectations, including software requirements (aligned to system requirements), prior to execution by a developer.
- Including both a chief systems engineer and a software architect on the staff.
- Addressing systems engineering and software metrics within a Risk Management Plan correlated to the Program Strategic Plan.

PEO IEW&S executive Edward T. Bair has taken (and is currently planning) specific actions to mitigate software system acquisition risks to include:

- PEO IEW&S Software Education Policy: Defining program software engineering education requirements for the Army’s PEO IEW&S Software Intensive System Programs. The plan for 2005 includes software acquisition education at the executive, program management and staff levels.

- PEO IEW&S Software System Acquisition Metrics Policy: Mandating program metrics to give evidence of critical insight and visibility into software system acquisition lifecycle processes and to demonstrate more effective and efficient software system acquisition lifecycle management. Relate to program probability of success.
- PEO IEW&S Software Acquisition Improvement Plan (SAIP): Guiding the PEO IEW&S Project Management Offices to promote focused and coordinated improvement in the fielding of software-intensive systems. ASSIP member organizations have developed a coordinated SAIP per FY05 ASSIP Master Plan signed by Gen. Yakovac on Oct. 1, 2004.

The PEO IEW&S continues to emphasize its goal to improve its software acquisition performance by continuing to focus resources on software system lifecycle policy, education, communicating implementation guidance, and expanding its software acquisition support infrastructure.



For more information on the Army Strategic Software Improvement Program, contact James Wessel, jwessel@mitre.org, 732-578-6469.

Guide to the Software Engineering Body of Knowledge

After more than 35 years of pursuing “software engineering” as an ideal, there is unmistakable evidence that it is finally emerging as a professional engineering discipline. Many nations (and the state of Texas) are chartering, licensing, certifying, or otherwise recognizing individuals as software engineers. A few universities now offer undergraduate degrees in software engineering, and the two major professional societies, the Association of Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE), have recently completed a joint software engineering curriculum.

All of these results are implicitly founded on the premise that an identifiable body of knowledge exists in the literature. The IEEE Computer Society’s Guide to the Software Engineering Body of Knowledge (SWEBOK) provides a consensually validated characterization of the software engineering discipline and provides a topical access to its Body of Knowledge. The about-to-be-published 2004 version of the guide* is an extensive update of the Trial Version published in 2001.

The Body of Knowledge is subdivided into ten Knowledge Areas plus an additional chapter providing an overview of strongly related disciplines. The text of each Knowledge Area provides a topical overview of the subject area, permitting readers to rapidly find their way to subjects of interest. When the subject is found, the reader is referred to key papers or book chapters selected because they succinctly present the knowledge.

The ten knowledge areas are:

- Software requirements
- Software design
- Software construction
- Software testing
- Software maintenance
- Software configuration management
- Software engineering management
- Software engineering process
- Software engineering tools and methods
- Software quality

Eight related disciplines are described in overview:

- Computer engineering
- Computer science
- Management
- Mathematics
- Project management
- Quality management
- Software ergonomics
- Systems engineering

Many topics of information technology (for example, programming languages, relational databases, and networks) are not represented in the guide. This is a consequence of the engineering approach. In all engineering fields, it is recognized that specific technologies are replaced much more rapidly than the engineering workforce. Engineers must be equipped with the basic knowledge that supports the appropriate selection of technologies.

Even within the confines of the software engineering discipline, the guide does not characterize all knowledge. Instead, it describes that subset of software engineering knowledge that is generally accepted as useful in most projects most of the time. In terms of level of detail, the guide is designed to address software engineers with undergraduate degrees and four years of practical experience.

The content of the guide was validated by an extensive consensus formation process that spanned five years. Nearly 10,000 comments were provided by more than 500 reviewers from 42 nations. All comments were disposed by associate editors for each knowledge area who worked under the direction of the overall editorial team.

Although the SWEBOK guide is a project of the IEEE Computer Society, MITRE made important contributions to the project. Jim Moore served as one of the Executive Editors with overall responsibility for the consensus formation process. MITRE provided support to the project by joining its Industrial Advisory Board—Chuck Howell (Director of Software Systems Engineering, W900 - Center for Innovative Computing and Informatics) was our representative. Terry Bollinger (Principal Engineer, Software Systems Engineering) served as the first editor for the Knowledge Area on software construction.



For more information on SWEBOK, contact Jim Moore, moorej@mitre.org, 703-983-7396.

* The IEEE Computer Society’s Guide to the Software Engineering Body of Knowledge is available at <http://www.swebok.org>.

Design Patterns: A Pattern for Strategic Advantage

People who create complex systems must have not only practices that aid the initial deployment of these systems, but more important, a design process that inherently supports constant evolution, which usually puts significant resources (function, cost, and schedule) at risk. In many fields, including software, pattern-based practices have been applied to the creation and evolution of complex systems. In DoD systems, it is not enough to be competitive—solutions must be superior to the competition. Design patterns support this imperative.

A pattern is a proven methodology for helping to create superior complex systems. This approach is based on the systematic reuse of knowledge. In creating a capability, the knowledge gained (not the transient data) is our most important asset. Unfortunately that knowledge from previous endeavors is rarely captured, organized, or reused for strategic advantage. We cannot be superior if each capability is rediscovered, refined, and validated at the cost of new capabilities that could provide competitive advantage. Patterns provide a strategy for reuse of strategic knowledge assets.

The format of a pattern typically includes a unique name, the context giving rise to the problem, a description of the forces that define the problem, the solution that resolves the problem, and the consequences, that

is, the benefits and limitations. A good pattern is very brief (two to four pages), concise in its language, abstracted from extraneous details for reuse, and validated by review and reference implementations. A library of patterns becomes a strategic asset to achieve competitive advantage because it helps us efficiently create and

In DoD systems, it is not enough to be competitive—solutions must be superior to the competition. Design patterns support this imperative.

evolve enterprise-level software systems.

Patterns have been applied to building design, object-oriented software, and most recently to the software services that integrate complex systems that define the Air Force enterprise. As the cornerstone of a Service Oriented Architecture (SOA), patterns are used in the context of a complete SOA framework, which should include a

service life-cycle model for accountability, formal service definitions of production services, and an overall service design and management strategy. Pattern-based services are a unit of currency for the SOA-based enterprise.



For more information on Design Patterns, contact Robert O. Wilson, bwilson@mitre.org, 781-271-4867.

Highlights of the next issue of Collaborations ...

WHAT'S NEXT?

In our next issue, we will look again at complex-system engineering. Complex-system engineering has become a topic for research, innovation, and improvement in industry and academia. What characterizes a complex system? Is engineering a complex system more difficult? Can traditional system engineering practices be applied to complex systems? Can such a system even be engineered? We ask people inside and outside MITRE for their perspectives on this subject.

WHAT DO YOU THINK?

We welcome feedback from our readers. Do you agree or disagree with what is in the newsletter? Do you have information you would like to add? We want to publish your responses to share with our other readers. Send email to sepo@mitre.org with your suggestions.



Who We Are

The MITRE Systems Engineering Process Office (SEPO) is a nexus for systems engineering information and activity at MITRE. Our team brings together useful systems engineering resources, provides guidance on systems engineering processes, and participates in systems engineering activities throughout The MITRE Corporation.

Systems engineering resources are available through the SEPO Library, which contains a broad spectrum of information and knowledge to help you on such topics as acquisition, systems engineering, software engineering, decision support, and process management.

Systems engineering expertise is available through the SEPO Technical HOTline. Emails sent to the HOTline reach multiple subject matter experts, who can provide answers to your questions, connections to other experts on the subject here at MITRE, or contact with other people who are working on the same problem.

We offer systems engineering guidance through our SEPO Toolkits. Toolkits are available online or on CD and include many topics on the system engineering process. For guidance in another area, such as Software Engineering, Acquisition, the Capability Maturity Model - Integrated (CMMI) Process, or sponsor-specific systems engineering areas, please contact our team.

Collaborations is a publication of SEPO. For additional information, please contact:

Brian E. White, Ph.D., SEPO Director
781-271-8218; Email; SEPO@mitre.org

This newsletter is a publication of the Systems Engineering Process Office (SEPO). Additional copies are available.

The MITRE Corporation, 202 Burlington Road, Bedford, MA 01730-1420, and 7515 Colshire Drive, McLean, VA 22102-7508

www.mitre.org/work/sepo/

Judith A. Clapp, editor
David Cleary, editor
Lori Fermano-Pass, designer

©The MITRE Corporation, 2005
Approved for Public Release;
Distribution Unlimited.
Case Number 05-0160