A Lessons Learned Software System

Scott Hawker
University of Alabama
hawker@cs.ua.edu
205-348-1667

Supported by the NASA Technical Standards Program Office
http://standards.nasa.gov

1. Introduction

NASA, like any learning organization, must capture its experience and learn from it. Lessons learned are crucial to the agency’s continued success. However, the GAO-02-195 report called for significant improvements in the way NASA collects and shares lessons learned information.

This document describes a software system to support the capture, approval, and communication of lessons learned at NASA. The report defines the system context, outlines the requirements, and sketches an architecture for the system.

2. System Context

The proposed system is a portion of a solution to the needs and approach outlined in two proposals to NASA Headquarters:

- “Effective Use of Lessons Learned: A Key Element for Mission Success,” submitted to the Office of the NASA Chief Engineer (Code AE) by the NASA Technical Standards Program Office on April 21, 2003,

As Figure 1 illustrates, lessons learned are generated while executing projects and programs. After acquiring and approving the lessons learned, they are available for use in later projects and programs. The lessons also serve as a source for new and updated standards and training materials. The proposed software system facilitates the acquisition of lessons from projects, the performance of the approval process, the management of lessons, and the communication of lessons to projects and programs.
Lessons learned are captured and used in the context of program/project execution. The performance of program and project management processes and systems engineering tasks trigger tasks to acquire or use lessons. Figure 2 illustrates that the lessons learned process must be coupled with the program/project management process and the systems engineering process that produce aerospace products and capabilities. For example, the beginning of an engineering phase or task should trigger the review of relevant lessons learned, and the passing of a management decision gate or a project review should trigger the capture of new lessons learned.
Figure 2. Lessons learned process in the context of the systems engineering process and the program/project management process

The program/project management process is defined currently by NPG 7120.5B (NASA Program and Project Management Processes and Requirements), and the systems engineering process is defined currently by SP-6105 (NASA Systems Engineering Handbook) and the evolving draft NPG 71xx.x (Systems Engineering Processes and Requirements). All of these processes require the application of and contribution to the lessons learned body of knowledge. The proposed Lessons Learned system should couple the workflow of lessons learned (LL) tasks with the workflows of other tasks identified in the program/project management and systems engineering processes that produce aerospace products and capabilities (PAPAC). That is, the LL processes should be an integral, closed-loop part of the PAPAC processes. Figure 3 illustrates this relationship.
Figure 3. Lessons Learned processes as an integrated, closed-loop part of processes to produce aerospace products and capabilities

Figure 4 shows the Lessons Learned Software System as part of a broader software and human system. The broader system has the following tasks:

- submit new lessons
- approve lessons
- manage lessons
- find and apply lessons
- perform tasks to produce aerospace products and capabilities
  - system engineering process tasks
  - program and project management process tasks

Figure 4. The Lessons Learned Software System as part of a broader system

The broader system has the following human elements and responsibilities with respect to the lessons learned process:

- task performer (contractor and/or civil servant)
  - submit lesson
• Discipline working group
  o review and approve submitted lessons
  o review and adopt existing (legacy) lessons
  o edit, remove, merge lessons
  o revalidate lessons
  o define discipline scope
  o define discipline ontology
• LL working groups
  o push existing and new lessons to users
  o facilitate lesson use
  o facilitate lesson gathering
  o gather submission and use metrics
  o champion LL process (awareness campaign, etc.)
• LL process facilitators
  o define and improve process
  o schedule and facilitate group meetings
  o track actions
• LL system trainers
  o train on how to use LL systems and processes
• LL software system developers
  o adapt existing or develop new software system
  o maintain and enhance system
  o incorporate existing (legacy) LL systems
  o administer system

3. Lessons Learned Software System Requirements
The Lessons Learned Software System has the following overall responsibilities:
• Store and display lessons learned
• Integrate existing (legacy) lessons learned systems into what the user perceives as a single, integrated system
• Accept new lessons learned
• Provide search and retrieval of lessons relevant to a given program/project management or engineering task
• Facilitate the following human workflows
  o Document and submit a lesson
  o Review and approve a lesson
    • with modification cycles, as necessary
  o Add approved lesson to system
  o Find and use lessons
  o Perform program/project management and system engineering tasks
    • Facilitate the application and contribution of lessons within the context of these tasks
From these responsibilities and based on similar systems, we derive the following set of requirements for the Lessons Learned Software System.

**Store and Display New and Legacy Lessons**
1. The system shall store lessons learned
2. The system shall incorporate, where feasible, existing lessons learned that are stored in separate, legacy databases (relational or flat files, for example)
3. The system shall allow the reading and display of existing lessons
   3.1. Lessons stored in separate, legacy databases should be available from the same user interface as lessons stored in the current system
   3.2. Lessons stored in separate, legacy databases should be displayed with the same fields and content in which they were captured (preserve presentation, where feasible)
4. The display of lessons shall be via a web browser
   4.1. Question: Allowable limitations on which versions of which browsers

**Organize and Search Lessons**
5. The system shall provide keyword-based search for lessons
   5.1. The system should provide so-called advanced search options
      5.1.1. Boolean (AND, OR, NOT) combinations of words
      5.1.2. Co-located words (phrases, words near other words)
      5.1.3. Word stemming
6. The system shall support organizing lessons in various taxonomic (category) structures
   6.1. The system shall support categorization of lessons using the technical disciplines defined by the NASA Technical Standards Program
   6.2. The system shall support classification of lessons as either technical or programmatic
   6.3. The system shall be extendable to support additional category structures
6.4. A given lesson can be categorized using zero or more category structures (categorization is optional; a lesson may categorized using multiple category dimensions)
   6.4.1. The specification of a given category dimension should indicate whether a given lesson can appear in none, exactly one, or more than one category in that dimension
7. The system shall support searching for lessons using taxonomic structures (drill-down)

**Capture New Lessons**
8. The system shall support the capture of new lessons
9. A lesson shall include at least the following elements
   9.1. Lesson title
   9.2. Lesson date (the date the lesson was learned, not the date the lesson was documented)
   9.3. Lesson summary (one sentence or one paragraph description of the lesson)
9.4. Lesson description (detailed description of the lesson, background, recommendations, etc.)
9.5. Author full name
9.6. Author email address
9.7. Author phone number
10. The system shall allow authorized users to define additional mandatory or optional lesson elements
    • Design note: The conceived architecture uses XML Schemas to define lesson content. A lesson must conform to the XML Schema corresponding to the minimal elements listed above in requirement 9 (the “base schema”). The lesson may also conform to an XML Schema that extends the base schema with additional elements or with constraints on base schema elements. The extending schema must be approved, the schema must be defined in a namespace managed by the system, and the system must have an XML Stylesheet for displaying lessons that conform to the extending schema.
11. The system shall allow a lesson to include a URI (web link) to referenced information
11.1. The referenced information should be stored in the new system to assure that the information is always available (avoid dead links or uncontrolled targets)
12. The system shall support associating zero or more files with a lesson (to provide additional information about the lesson)
12.1. The file format should be limited to standard MIME types that are supported in most web browsers and NASA desktop installations (Adobe PDF, JPEG and GIF images, Microsoft Office applications (Word, PowerPoint, Excel), and to-be-defined sound, video and multimedia formats)

Manage Lessons
13. The system shall allow an authorized user to delete a lesson
14. The system shall allow an authorized user to edit a lesson

User Profiles and Lesson “Push”
15. The system shall allow users to register interest in lessons
15.1. The system shall allow the user to indicate interest using specific keywords and phrases
15.1.1. The system should support specification using the “advanced search” features including Boolean (AND, OR, NOT), phrases, etc.
15.2. The system shall allow the user to indicate interest using lesson categories
15.3. The system shall allow the user to indicate interest by naming their current program/project management task or engineering task
16. The system shall notify registered users when a new lesson is approved that meets their profile
16.1. The notification shall be via email
16.2. The notification may be via additional alternate means to-be-defined

Workflow Support
17. The system shall provide a workflow utility to facilitate the approval of entered lessons
17.1. The approval process is as defined in “Effective Use of Lessons Learned: A Key Element for Mission Success,” submitted to the Office of the NASA Chief Engineer (Code AE) by the NASA Technical Standards Program Office on April 21, 2003

17.1.1. Or other process to be defined

18. The system may provide a workflow utility to facilitate the capture and application of lessons as part of performing the System Engineering Process (NPG 71xx.x or SP-6105).

19. The system may provide a workflow utility to facilitate the capture and application of lessons as part of performing the Program/Project Management Process (NPG 7120.5B).

20. The system should be extensible to allow authorized users to insert a workflow utility to facilitate the capture and application of lessons as part of performing some defined process (for example, as part of the execution of a local organizational work instruction)

**Metrics**

21. The system shall provide metrics to be defined

**Non-Functional Requirements**

22. The system shall conform to Section 508 accessibility requirements

23. The system shall be available 24x7x365

24. The system shall provide facilities to back up and restore the lessons

25. The system shall support a to-be-defined number of concurrent users

26. The system shall support at least 100,000 individual lessons

27. The system shall respond to access requests within five seconds

28. The system shall respond to search requests within ten seconds

28.1. The system shall provide an indication of progress or activity in search requests

**4. Architecture Concepts**

This section presents architecture concepts toward a possible system design that realizes the requirements for the Lessons Learned Software System. The conceived architecture, shown in Figure 5, combines a number of off-the-shelf components:

- lesson repositories
  - a native XML database
  - legacy repositories
  - file system to store files associated with lessons

- a user, task, and lesson metadata repository

- a search engine

- a workflow engine

- a web server

- applications running on an application server

- a web browser
The next sections detail the architecture to address lesson manipulation, search, workflow, and other functions.

Figure 5. Architecture overview

4.1. Lesson Manipulation
The proposed lesson creation and manipulation architecture, shown in Figure 6, uses XML to represent lessons, and it uses Java applications to create and manipulate lessons. A lesson is an XML document stored in a native XML database (a database that stores XML documents in their native format, rather than converting them to relational tables or some other storage model). The lesson document conforms to an XML schema that defines the allowed fields and structure of the document. A base schema defines the minimal required elements in a lesson. Extensions to this base schema allow alternate lesson fields to be included in an alternate lesson format. XML stylesheets for a schema define how to display XML documents that conform to the schema. Stylesheets are defined using the eXtensible Stylesheet Language (XSL) or Cascading Stylesheets (CSS). By defining a family of XML schemas and corresponding XML stylesheets, the system can accommodate multiple lesson learned formats presented in a consistent, uniform way. To assure stability, all XML schemas and stylesheets should be defined in XML namespaces under control of the Lessons Learned Software System.
Java applications, packaged and launched as Java Servlets, use DOM, SAX, and other protocol standards to create, validate, and manipulate XML documents. New lessons (as opposed to existing, legacy lessons) are stored in a native XML database. Access to the database is via the XML:DB protocol. Files associated with lessons are stored in the local file system and referenced as URIs (web links). From within the XML documents, additional semantics of the associated file are defined using the XLink standard.

Note, also, that Witango 5 has built-in support for creating and manipulating XML documents, so some functionality could be implemented directly in Witango rather than in the Java applications.

Legacy lessons that are stored in databases can be made part of the system if an XML wrapper can be defined. This involves defining an XML schema and stylesheet for the lesson (hopefully, extending the base lesson schema) and implementing the DOM or SAX protocol for the lesson. For example, there are implementations that present ODBC-compliant data tables as DOM objects.
4.2. Lesson Search

Figure 7 suggests an architecture to provide search functionality. Commercial search engines, such as Verity Ultraseek, have built-in support for searching web pages, file systems, XML documents, and ODBC-compliant databases. The search engine can be integrated into the Lessons Learned Software System using Java APIs. These APIs could be called directly from a web application server, such as Witango, or indirectly through a Java servlet application.

Although open source and other freely available search engines are available, the advanced search capabilities of commercial search engines is of value. An engine such as Verity Ultraseek with features to search files, web pages, relational databases, and XML documents is available for $10k – $15k (ballpark estimates).

To facilitate associating lessons with users, the proposed architecture defines metadata to characterize lesson content, user preferences, and user task needs. The metadata is captured as XML documents and stored in the native XML database, as Figure 6 shows. As Figure 8 suggests, characterizing the lesson, user, and task in terms of a domain knowledge model will aid searching and matching lessons with users. See the Standards Advisor Vision and the MetaMatch description in the Standards Advisor status report for more on these concepts [see http://cs.ua.edu/SEL/StandardsAdvisor/].
4.3. Workflow

The workflows for submitting and approving lessons are simple enough that a Witango application or Java application could implement the workflow process. However, to enable the creation of workflows to support other processes, such as system engineering or project management, the architecture includes a configurable workflow engine, as shown in Figure 9. The approach to integrating the workflow engine is similar to the integration of a search engine. A workflow engine can be integrated into the Lessons Learned Software System using Java APIs or other service invocation functionality provided by the engine. These APIs could be called directly from a web application server, such as Witango, or indirectly through a Java servlet application. Most workflow engines provide easy integration with external systems, such as email to notify users of work tasks or system data access from relational databases or XML documents.
There are numerous commercial and open source workflow engines available. Commercial engines (also known as business process management systems) are available as part of larger enterprise management solutions from big players such as SAP, IBM, Fujitsu, Microsoft, and Oracle. Medium-sized players such as Staffware, FileNet, and TeamWare also offer solutions. Pricing information for these products has not yet been obtained.

Since the needs of the Lessons Learned Software System are modest, an open source solution may be acceptable. Examples include

- OpenFlow (http://www.openflow.it/)
- WERKFLOW (http://werkflow.codehaus.org/)
- OBE (http://www.openbusinessengine.org/)
- jBpm (http://jbpm.org/)

Unfortunately, these open source solutions are immature, and some are still in development.

It may make sense to “hard code” the lesson approval process (rather than having a configurable workflow engine) in early implementations of the Lessons Learned Software System, then transition to a commercial or open source implementation at a later date. Alternatively, an early implementation based on a commercial solution could migrate to an open source solution, later. Conformance to standards such as those set by
the Workflow Management Coalition (http://www.wfmc.org/), ebXML (http://www.ebxml.org/), and BPMI (http://www.bpmi.org/) will help migration of workflow specifications to alternate workflow engines.