



Application-Specific Knowledge Bases

Dr. Babak Makkinejad
Electronic Data Systems

It is suggested that companies create and maintain a searchable knowledge base to capture specific issues that are encountered and resolved during commercial off-the-shelf and custom software development projects. Such a knowledge base could benefit the current development staff, the future sustainment staff, and the testing staff, hence facilitating further evolution of the system during its life cycle.

In this article, I will discuss an activity related to both commercial off-the-shelf (COTS) and custom software development that I believe is not often addressed in practice. This activity, often ignored, is the capture, sustainment, and viewing of application-specific knowledge that is generated and acquired by the development associated staff during the course of a development project for the benefit of the sustainment programmers and staff. I will also discuss enablers for addressing this shortcoming.

Undocumented, Poorly Documented, and Work-Around Features

Before proceeding further, I need to define what I mean by COTS and custom software. By COTS software development, I mean customization of COTS systems such as Microsoft Office Suite, Siebel Systems Customer Relationship Management software suite of products, SAP, TeamCenter Enterprise (aka Metaphase) from Unigraphics Solutions, and so on. By custom software development, I mean the development of a software system from scratch but admitting the incorporations of some third-party COTS products such as (class) libraries, application frameworks, and so on. I will note here that, as others have observed, the distinction between COTS and custom is not rigid; a C language compiler and linker may be viewed as the simplest (lowest) COTS application.

Regardless of the rigor in the definition of COTS versus custom applications, it has been my observation that the development staff encounters and resolves a number of issues during the course of the development phase that are of great value to the sustainment phase of the system's life cycle. The development staff often takes advantage of undocumented, poorly documented, or defect work-around features involving the following:

1. COTS Application Programming Interfaces (API).
2. COTS Data Model.

3. Other (third-party) COTS API and Data Models.
4. Vendor-/Version-Specific Operating System.
5. Vendor-/Version-Specific Middleware (class libraries, frameworks, etc.).
6. Network Protocol and Environment.
7. Vendor-/Version-Specific Backend Database.

This is done to meet the business/technical requirements of the software system being constructed. Often, there is no other way than to use these features to meet the delivery deadlines of the system. And, contrary to a popular belief among development managers, COTS application development is also subject to the above considerations.

More crucially, the evolution of the ingredients that have gone into building the system, from software add-ons to the operating environment, may cause some or all of the undocumented features, poorly documented features, or work-around to no longer pertain to a new release of a particular building block of the system. Consequently, the system may undergo significant performance degradations or cease to properly function altogether. The existence of such a knowledge base will go a long way to expedite the resolution of such problems in production.

You must also note that it is most often the case that the knowledge of these features and development *shortcuts* becomes disseminated among the development staff as a sort of tribal knowledge. Unfortunately, this tribal knowledge is usually lost when the system is turned over from the development staff to the sustainment staff and the development tribe is reassigned, or dissolved (which is more likely the case). Thus, further evolution of the system during its sustainment phase could be compromised due to the unavailability of this knowledge base of tribal *know how*.

The usage of these undocumented, poorly documented, or defect work-around features is normally not captured in the

technical design documentation. The technical design documents are seldom revised during the course of a typical development project and often are not detailed enough to even provide space for capturing this type of knowledge.

In programming practice, some of this knowledge could be available in the form of source file comments. Even then we are facing the challenges that the individual developer may or may not have provided useful comments, or that the comments may or may not be relevant to the current code revision. Additionally, when these features are in different technical areas, for example API as opposed to data model, they cannot, even in principle, be captured as part of the code (the code-as-documentation crowd notwithstanding). In fact, in such cases, often different individuals or groups are leveraging these undocumented/poorly documented features and thus, are unaware of one another's work. This makes capturing this information in a common technical documentation and format more challenging, but at the same time, more crucial.

Application-Specific Knowledge Bases

It is suggested that an effort be made to develop a knowledge base for the system being built during the development phase of the system. This knowledge base should be able to capture the following:

1. Undocumented features used in developing the system.
2. Poorly documented features used in developing the system.
3. Defect work-around features used in developing the system.
4. Version/patch information of the building blocks of the system.
5. Implemented kludges.
6. Discussion of the technical reasons for using these features.
7. Things that could go wrong if these features cease to work in a future release.
8. Recommendations for replacing these

features in future releases.

9. Date and time stamp for all entries in the knowledge base.
10. Preferred methods for incorporating features used in developing the system.
11. Standards and guidelines that should be used to govern system development.

This type of knowledge base must be distinguished from a frequently asked questions or a technical design document. It should be thought of as an *undocumented corner* magazine column, a pitfalls list, or a compilation of *programmer's shortcuts*. It can be thought of as a small analogue of the large, product knowledge bases that vendors such as IBM, Oracle, Sun, Microsoft and others supply.

While such large knowledge bases are for commercial products, the knowledge bases discussed in this article pertain to specific applications produced for specific clients and hence the phrase: *application-specific knowledge bases*. And, in an analogous manner, these application-specific knowledge bases are conceived to be easily accessible and searchable.

An application-specific knowledge base is thus an enabler for expediting the resolution of defects and in assessing the risks involved in the evolution of specific systems as requirements, building blocks, and operating environment evolve over time. Both the technical development staff and the business staff will be consumers of this information for technical and business purposes.

This type of knowledge base must be distinguished from a defect tracking/help desk system. Although some or all of the issues and knowledge captured in application-specific knowledge bases may already exist in a defect tracking system, they are not there in a usable format. More information on this follows.

Challenges to Adoption

There are two challenges with realizing the above vision. Perhaps the most significant barrier to adoption of this practice as part of the software engineering process is the development staff buy-in. In many cases, members of the development staff are required to supply/update documentation that, in their minds, has no value. In fact, many development staff members consider much of the documentation effort misplaced and non-value-added.

Another major impediment to developer buy-in is the time factor; development staff is typically under so much time pressure that they cannot find time to create what they perceive as an additional system (the knowledge base) beyond the system they are funded and allocated to create.

Additionally, in many instances the development staff is not going to be around for the sustainment phase. It is a leadership challenge to motivate the development staff and inspire them to do a great job and to facilitate the activities of the sustainment staff.

Perhaps the best way to proceed is to periodically ask the development staff to supply a list of items that they feel will be important to know for the sustainment phase in free form. Then as part of the process of software documentation, one can compile these inputs into a knowledge base as part of the key deliverables of the system.

The other challenge is the packaging and delivery of this knowledge base. For the knowledge base to be useful, it must satisfy the following criteria:

“The technical design documents are seldom revised during the course of a typical development project and often are not detailed enough to even provide space for capturing this type [undocumented, poorly documented, or work-around] of knowledge.”

1. **Be easily accessible.** The user should be able to get to the knowledge base without having to navigate a hierarchy of network folders or Web-based pages. Nor should he or she have to look for a specific file among numerous electronic documents for the information that he needs.
2. **Be easily searchable.** The user should not have to use Global Regular Expression Parser or the Search feature of the operating system to look up the information for which he is searching.
3. **Be easily navigable.** The user should be able to easily and painlessly move from one item to the next relevant item.
4. **Be easily updateable.** The knowledge base should be easily updateable to

reflect the changes to the software, its building blocks, and its operating environment.

5. **Be portable.** The user should be able to view the information without having to be hooked to the enterprise network; it must be accessible in a disconnected mode from a portable computing device. This is an essential feature for certain class of applications that require on-site support.
6. **Be secure.** Unauthorized access to use the knowledge base for malicious purposes must be prevented.

One approach will be to utilize the same COTS tools that have been used for building, updating, and maintaining the software's help system to deliver such an application-specific knowledge base. These tools are often multi-platform, thus enabling the development of a knowledge base that can be made to satisfy all of the criteria above. These types of systems are easily searchable, often use hypertext to provide navigable links, can be edited by using a word processor, and are portable. Unfortunately, this is a heavyweight approach since it requires knowledge of the specific help system creation tool, and the effort itself will become part of the cost of the development.

The next best candidate for the deployment of such a knowledge base will be to leverage an existing defect tracking/help desk system by augmenting it with the development staff's issues and resolutions. With this approach you have the added challenge of customizing the defect tracking system to distinguish among generic defects and the application-specific *gotchas* of the knowledge base. It must keep track of the requirements, building block revisions, patches, and changes to the operating environment that pertain to the items captured in the knowledge base.

Unfortunately, even though most tools are, in principle, capable of satisfying a number of the above criteria, they do not satisfy them all. Specifically, in the areas of navigability and portability, they leave much to be desired. While the navigability criterion may be addressed with the addition of *Google-like* features, the portability will always be an issue for most of these centralized systems.

The simplest approach, which is quite doable and lightweight to develop and deploy, is to create a hypertext markup language (HTML) document that will contain the knowledge base. In this approach, all that is needed is a text editor and staff who are knowledgeable in HTML; expensive development tools will not be required. This approach satisfies the first

five criteria above.

The three approaches above all require additional developer interactions that are specific to the knowledge base. They add to the development effort and cost. There is an alternative approach to developing the knowledge base that leverages a common activity that occurs during the course of a development project; that is, team members send one another e-mails discussing the problems they need to solve, how to work around limitations of the tools, etc. In fact, on the Internet, the archival and current material in technical discussion forums serves exactly the same purpose. I am suggesting archiving the e-mail exchanges of developers for the sustainment phase.

This developer e-mail base, plus the other documents that are created and assembled by the development team is, in effect, a knowledge base that can be a resource for sustainment staff. To leverage this naturally created knowledge base, you need a generic, straightforward method to capture this source of information at the end of a development project and make it available to and easily searchable by the sustainment team.

This would entail using the built-in archiving functions of the messaging server at project start-up, followed by the search capabilities of the e-mail client itself such as Microsoft Outlook on a *.pst* file. In fact, there are currently e-mail clients that support something akin to Google to search e-mails, for example, Bloomba at <www.statalabs.com>. In this manner, you may enable the sustainment staff to quick-

ly find information (or, at least, clues) about how specific problems were solved by the development staff.

The sixth criterion, the security requirement, is a challenge for all approaches. For those applications that will have their own security features, access to the knowledge base may be controlled by leveraging the application's own security features. In other cases, you must consider the details of secure access to the knowledge base and assess the risks involved in unauthorized access to this data. While 100 percent security is not even theoretically achievable, a judicious approach to access control and distribution lists should largely mitigate the security concerns related to the knowledge base.

Conclusion

I believe it a good idea to canvas the development staff for issues that pertain to using undocumented, poorly documented, defect work-around, kludges, and gotchas that have been encountered and/or leveraged during the course of system construction for both COTS and custom application. I further conceive of the utility of compiling that input into an application-specific knowledge base for consumption by the technical and business staff during the sustainment phase of the system.

Although there are multiple cost-effective techniques for packaging and enabling such a knowledge base based on available commercial tools, I favor capturing e-mail exchanges as the most lightweight method. Since the amount of data in the knowledge

base is necessarily limited to the development phase, *key* word searches will not be as tedious as on the Internet since the number of hits would be either small or none at all.

Finally, I believe that human factor issues are the greatest barrier to developing and deploying such a system. I respectfully urge the project leadership in the information technology industry to make a concerted effort to supply the necessary positive motivations for this effort to become practicable. We owe this to those who come after us to sustain the systems that we are producing today. ♦

About the Author



Babak Makkinejad, Ph.D., is a consultant with Electronic Data Systems. He has worked in the areas of computational physics, computer graphics, image processing, and enterprise software development. Makkinejad has a doctorate in theoretical physics from the University of Michigan in Ann Arbor.

Electronic Data Systems

5555 New King ST

Troy, MI 48098

Phone: (248) 696-2311

Fax: (248) 696-2590

E-mail: babak.makkinejad@eds.com

MORE ONLINE FROM CROSSTALK

CROSSTALK is pleased to bring you additional articles with full text at <www.hill.af.mil/crosstalk/2005/06/index.html>.

Knowledge Management and Process Improvement: A Union of Two Disciplines

*Gregory D. Burke
Federal Aviation Administration
William H. Howard
Northrop Grumman Mission Systems*

The experience at the Federal Aviation Administration (FAA) shows that process improvement and knowledge management complement each other well. Process improvement helps the organization increase its effectiveness through continuous examination with a view to doing things better. Once processes are documented, roles and responsibilities are readily identified and associated activities are performed. Legacy processes are modified to reflect organizational changes. Knowledge management facilitates communication among organizations, increasing information sharing and utilizing process documentation. This informa-

tion sharing promotes organizational unity and allows FAA headquarters and regional operations to function efficiently.

Connecting Earned Value to the Schedule

*Walt Lipke
Tinker Air Force Base*

For project cost, analysts can predict the final value with some confidence using the Independent Estimate at Completion (IEAC) formulas from Earned Value Management (EVM). However, EVM does not provide IEAC-like formulas by which to predict the final duration of a project; many express the opinion that schedule information derived from EVM is of little value. This article discusses the problem and develops a methodology for calculating the predicted project duration using EVM data. The methodology uses the concept of Earned Schedule and introduces an additional measure required for the calculation.