Evaluating Open Source Software for Health Information Exchange

HIMSS Healthcare Information Exchange
Open Source Task Force White Paper

June 2008

Table of Contents

Introduction

A recent review of developing Health Information Exchange (HIE) projects focused on governance and operational models,[1] but was silent on technical approaches. The majority of projects examined in a recent review of projects that had been established, including ones that had failed,[2] also did not review any of the software technologies that had been used to deploy the projects. For instance, the suggested reason for failure for the Santa Barbara County Clinical Data Exchange was "failure to obtain sufficient participation from local stakeholders," but the authors did not explore how the technical approach itself might have had an influence on the outcome of the project. While a strong organizational structure is essential for an effective HIO (Health Information Organization), choosing the right technology is also critical. The time is right to explore the suitability of open source software technology for HIE projects.

In the ensuing discussion, the terms HIE and HIO will be used, recognizing that the Regional Health Information Organization (RHIO) is one popular form of HIO. The use of these terms in this report is in keeping with a recent publication from the U.S. Department of Health and Human Services (HHS).[3] Whether the subject is a local exchange, regional exchange, state-based exchange, or a node of the National Health Information Network (NHIN), a careful study of the architecture and technology underpinnings is critical to understanding the organization's successes and failures.

A small minority of HIE projects have used the term "open source" to describe some component of the technology used in the project. Although there have been occasional reviews of open source software's applicability for healthcare information technology,[4] they have not specifically explored the potential for use of open source software for HIE implementations.[5] In this paper, a number of issues regarding the use of open source software in healthcare will be explored, but with a particular emphasis on considerations for HIE projects. First, how the word "open" has been used—in "open" standards and in "open" source—will be addressed.

---

[1] Holland Marc. It's Spring and the HIEs are Blooming. *Health Industry Insights* #H1211724; April, 2008.
[2] Adler-Milstein Julia., et al. The State of Regional Health Information Organizations: Current Activities and Financing. December 11, 2007. Available at http://content.healthaffairs.org/cgi/content/full/hlthaff.27.1.w60/DC1
[3] Defining Key Health Information Technology Terms. A report from the National Alliance for Health Information Technology to the Office of the National Coordinator for Health Information Technology; April 28, 2008.
[4] Goulde Michael, et al. Open Source Software: a Primer for Health Care Leaders. California Health Care Foundation; March, 2006.
[5] Fauss Samuel, Sujansky Walter. Open Source EHR Systems for Ambulatory Care: A Market Assessment. California Health Care Foundation; January, 2008.

Open Standards

The development of standards for software in healthcare has been an essential step for creating architectures for rational health information exchange.[6] Healthcare information technology (IT) has implemented a number of standards that are used in other industries (such as XML for message formats and the ISO 17799 Security Standards and NIST-800 Series Security Framework which have largely been incorporated into HIPAA). In addition, a number of standards have been developed which are used almost exclusively by healthcare. Healthcare IT standards include messaging standards such as the various versions of HL7, as well as terminology standards such as SNOMED and LOINC.

One criterion that is often used to evaluate a particular standard is its "openness." In this case, "open" refers both to the breadth of the community involved in developing the particular standard as well as to the transparency of the development process. Transparency (publicly scheduled meetings and calls, regular correspondence, predictable cycles of review, widely disseminated proceedings and meeting minutes, etc.) encourages and facilitates community involvement. Broader community involvement predicts broader usage of a standard. Standards that are broadly used and understood across a community foster interoperability, which is a cornerstone of rational health information exchange.

The business of creating Standards Development Organizations (SDOs) requires funding. SDOs are typically funded in a variety of ways including membership dues, parent organization support, grants, and licensing. While certain standards, such as CPT-4, operate in a business model that depends on comparatively high licensing fees from organizations wishing to implement the standard, there has been significant movement toward reducing the license costs for other standards, most notably SNOMED. That does not mean that the costs for maintaining SNOMED have disappeared; rather, those costs have been absorbed first by National Library of Medicine and then by International Health Terminology Standards Development Organization (IHTSDO.)

Open standards can be used by any software development organization and by any customer of a software development organization so long as both organizations adhere to that particular standard's licensing stipulations. It follows, then, that any standard can be used by both proprietary and open source software development organizations. It also follows that if any software development organization does not incorporate those open standards that are required for interoperability, the products from that organization are not likely to be very useful for rational health information exchange projects.

Simply put, interoperability is the ability of two or more systems (human or machine) to exchange information based on previously agreed upon structure and/or meaning. Unfortunately, the term is confusingly applied to several situations:

[6] Uniform Data Standards for Patient Medical Record Information. National Committee on Vital and Health Statistics Report to the Secretary of U.S. Department of Health and Human Services; July 6, 2000.

- Syntactic interoperability is a condition in which structures are exchangeable but there is no guarantee that meaning will be consistent across system boundaries. Earlier versions of HL7 carefully specified the syntax (structure) of data elements and messages but did not ensure consistent semantics of elements across messages.
- Semantic interoperability refers to the exchange of meaning between systems. Humans are adept at exchanging meaning (understanding). Machines, however, have a very limited capacity unless the exact meaning of each piece of data is unambiguously defined. Increasingly, healthcare IT projects require semantic interoperability. This feature is readily demonstrable between humans using email and the Internet, but is much more difficult when machine processing of data prior to human interpretation is required (e.g., machine-assisted decision support).
- Computable semantic interoperability (CSI) refers to the exchange of consistent and unambiguous meaning between two or more machines. This does not mean that all machines must process a given piece of data identically, but rather that all machines receiving a given piece of data interpret its meaning to be the same.[7]

Since achieving effective HIE is dependent on the shared use of clinical information, standards for ensuring semantic interoperability are essential; otherwise, a hospital system could be exchanging information with a physician office system but neither system recognizes what is being transmitted.

Open Source Terms

Just as "interoperability" has come to mean different things to different people, "open source" has a wide range of meanings in two divergent areas:

- "Open source" software development as an engineering process; and
- Licensing costs for "open source" intellectual property.

Below, open source licensing costs are discussed, and in the following section the open source software development process is reviewed. The key point is that "open source" is a broadly inclusive term that by itself lacks specificity to the extent that two nominally open source software packages may be completely incompatible in a given situation based on a various licensing and technical conflicts.

Open Source Licensing

There are many different types of open source licenses. Appendix A provides an overview of open source licensing prepared by the HIMSS Ambulatory Information Systems Open Source Work Group. Some licenses are incompatible with each other. The Open Source Initiative[8] website maintains a list of 70 separate open source licenses that have each been approved by a formal (and open) process.

[7] Jones TM, Mead CN. The Architecture of Sharing. *Health Informatics*. Nov, 2005
[8] http://www.opensource.org/licenses/alphabetical

Public Domain Software

Public domain software is software that is not copyrighted. By itself, public domain software is not open source (although it may or may not be developed under an open source process). A public domain software package may have one or more dependencies on a proprietary software environment in order for it to be effectively used. Public domain software is generally available at no cost or minimal cost (i.e., $5).

Free Software (No Cost to User)

Some open source software is released for free (no cost to user). Anyone can download it and, providing that one has the appropriate hardware and operating system (if required), the software can be used. OpenOffice.org[9] is an example of free software released under an open source license. Free software can be proprietary as well as open source. One cannot assume that free (no cost to user) software can be copied, modified, or distributed, unless it is released under an open source license. One cannot assume that source code is available for free (no cost to user) software.

Free Software (As in Freedom)

Free (as in freedom) software means that source code is available[10] and that a developer can copy, modify, and/or distribute the software. Its use/distribution may require a fee. It is this use of "free" that best captures "open source."

Open Source Software Development

As part of a healthcare software contract, a funding organization may stipulate that code developed for the project is made available to many other users without end user license fees. In some cases, this stipulation is met by creating the required software in an open source development environment and releasing it under an open source licensing model. More commonly, the code is developed in relative obscurity and then released for broader use when the project is complete. Because the code has been developed for a specific project, it may not be broadly applicable. Moreover, because the people who developed the code may consider their jobs to be finished when the code has been delivered to the funding organization, there may be little in the way of ongoing support available to other potential users outside of the organization that funded the project.

Occasionally, an entrepreneurial group will decide to "productize" the code that had been developed for a project. That group will then attempt to establish a business that allows the group to garner revenue for supporting the code outside its original project use and to charge consulting fees for extending the code (hopefully in a carefully governed way) for

[9] http://www.openoffice.org
[10] The most widely used collaborative site for housing open source software is SourceForge.

additional functionality. While this is a commendable activity,[11] there are hazards. If the software architecture/design has been created for a specific project without a broader view from the outset, a great deal of re-work will need to be done. Moreover, many such projects are released to the open source community upon completion rather than being released incrementally. This means that community input (along with the attendant scrutiny to quality) will be far more difficult to incorporate. Finally, many such projects may have incorporated proprietary code or dependencies on proprietary code that were not problematic for the original funding organization but that quickly become issues for other organizations that wish to use the code from the project.[12]

Leveraging Open Source Communities

Open source software development is a discipline. Creating new open source code can be very much like performing on the high wires without a net. The creator(s) of the initial kernel in the project or product places the code on SourceForge (or a similarly available site) and then seeks all comment and criticisms. Bugs are vigorously discussed and fixed in the open, lending an unusual degree of transparency to the development process. This degree of community engagement takes time, and may give the appearance that open source development is more time consuming than traditional code development. The trade-off of the open source process is a higher degree of software quality. Detecting bugs before the software is placed into actual use is just one reason why good open source software can be superior to proprietary software.

One way to accelerate an open source development process is to leverage existing open source components and the communities that support them. Leveraging a collection of open source communities rapidly increases the number of eyes examining development. Moreover, using robust open source components will likely increase the number of engineers who may be called upon to support a complex open source project, such as would be entailed for HIE. For instance, there are more engineers who are familiar with open source databases such as PostgreSQL and MySQL than with less widely deployed open source databases. The same comment can be made about the use of Java versus less widely used programming languages.

"Governed" open source projects/products are those in which the addition of code to the core code line is tightly controlled. For instance, while many engineers have made changes to the core Linux code, the code from a handful of those engineers (the "committers") actually becomes part of the core Linux code. Any code changes that are not committed to the core Linux product represent forks in the code that are not necessarily supported by the mainstream developers.

---

[11] This sort of evolution should be familiar to workers in healthcare IT. Many products in use today began life as projects in academic or governmental settings.

[12] The California HealthCare Foundation faced this dilemma in 2006 when it sought to release the proprietary code created for the Santa Barbara County Care Data Exchange to the open source community. To convert the software to an open source license required identifying and removing all proprietary portions of the software code that would be incompatible with the open source license under which the code was to be released. http://www.chcf.org/topics/view.cfm?itemID=132846

"Ungoverned" open source projects may encourage broad and relatively indiscriminate community contributions without the sense that a main code stream is being developed. Major support organizations such as Red Hat are more likely to support open source projects/products that are carefully governed. We believe that it is very important for organizations that are considering using open source software for health information exchanges to fully understand how the code is governed and how the code is supported for implementation at a customer site.

Open Source Products

If one considers all of the software in healthcare that has been labeled "open source," either mistakenly or accurately, one discovers relatively few "pedigreed" open source products. A *true* open source product is developed, from the onset, in a governed open source development environment using, wherever possible, tested open source components. An open source product is not developed for a specific customer but is developed to solve a set of problems faced by many healthcare organizations; it has broad applicability. Iterations of the code have been available for community review long before the product is considered to be generally available (GA) for use at customer sites.

An open source product should have not only a dedicated development team but should also have one or more dedicated support teams (development and support teams may be in different organizations). Organizations considering the use of open source software for healthcare information exchanges are advised to check for the availability of support services. It is critical for all HIOs to secure strong technical skills to implement enterprise level software. Interestingly, organizations that seek to provide commercial support services for an open source product as well as organizations actively implementing an open source product may attempt to identify potential members of their work force by investigating the community participants for the product. This is just another way in which developers can leverage their participation in open source development.

If the open source product is maintained on SourceForge, for example, one can assess the degree of community involvement from the activity posted on the product's SourceForge project site. The various forums for community participation should have many more participants than just the code committers. The forums should be sounding boards for high level clinical functionality comments as well as low level chatter about relatively arcane technical nuances. If a product fails to engage the community, the SourceForge forums will show little in the way of ongoing conversations; this can be a sign that the code is not being examined and may indicate that a valuable community oversight activity is failing to take place.

To further transparency, an open source product development organization should have a menu of white papers that are aimed at various levels of users (ranging from clinicians to engineers). In addition, the development organization should have taken pains to ensure ease of downloading by preparing various kits for trial use.

Open Source and Security

Since there is an admirable emphasis on protecting patient privacy in regards to healthcare information, managers need to be assured that an open source product implementation provides the level of security that is expected by consumers and clinicians. There is an unfortunate tendency to conflate security with secrecy. When such confusion reigns, then it is easy to carelessly assume that open source code must not provide the same degree of security as proprietary code. In other arenas, such as elections, where privacy and security are also very important, it has become apparent that using open source software is likely to ameliorate security concerns rather than increase them.

> "When I talked to Jennifer Brunner in October, she told me she wished all of Ohio's machines were "open source" – that is, run on computer code that is published publicly, for anyone to see. Only then, would voters trust it; and the scrutiny of thousands of computer scientists worldwide would ferret out any flaws and bugs."[13]

A recent study demonstrated that a substantial number of projects in the U.S. Department of Defense and in the Intelligence communities have been implemented using open source software and that security considerations were critical in making the choice.[14] If anything, use of open source software enhances security.

Open Source Business Model

Since an open source product traditionally does not require the purchase of a license, how do organizations "make money" from open source products? Since most companies that develop proprietary healthcare IT products derive their income from multiple activities, of which license fees are one, it should come as no surprise that commercial organizations can garner revenue from providing support services and custom code extensions with open source products. The sales/marketing cycle for an open source company may bear little resemblance to the cycle of those activities for a proprietary software company.[15] In many cases, the customers come to the commercial open source support organization after they have downloaded and "tried out" the open source code.

HIE Functionality Considerations

Over the past few years, it has become apparent that the functional requirements for HIE and for a hospital-based health information system are both overlapping and different. It has therefore proved to be challenging to transform any traditional clinical information systems into a foundation for HIE, although attempts have been made and are continuing

---

[13] Thompson Clive. Can you count on these machines? *New York Times Magazine*. January 6, 2008.
[14] http://www.federalopensourcealliance.com/#Study
[15] A very engaging look at the contrast can be found at http://www.linuxworld.com/news/2007/081607-matt-asay-interview.html?page=1

to be made. For instance, none of the four "original" NHIN prototypes employed a traditional clinical information system vendor in their architectural approach.

Other attempts have employed software that was not originally designed with healthcare in mind.

While some solution sets were built from the ground up to address the unique technical and operational functions of RHIO/HIE entities, others evolved from the expansion of the functionality of products that were originally targeted at related business functions such as workflow management, secure messaging, or Web portal development.[16]

For an organization responsible for HIE to be interested in an open source product, that product should have incorporated the core functionality of secure information exchange among the community stakeholders.

Because the desired functionality for HIE can be expected to change as new experience with this community-centered model is gained, the use of open source products is particularly attractive because extensions to the code can be proposed by a much larger community of developers and users. Moreover, if certain extensions developed for one organization prove to be useful, the community-focused nature of HIE is likely to support contributing that code to the open source product line (of course, the code committers would need to endorse the contribution).

Benchmarking

Successful HIE must be able to scale to a large number of patients (figures in the millions are not unusual). The easy availability of open source code should encourage benchmarking so that hardware determinations for HIE are less guesswork and more extrapolation. Simulations of data loads for a particular HIE project should be available before long-term commitments are made by the organization.

HIE Open Source Product Check List

Based on the discussion so far, a short check list can be created that will allow HIE project managers to screen available open source software for use.

1. Does the product incorporate the required level of open standards (messaging, vocabulary, information model) to be useful in HIE?
2. Does the product leverage robust open source components?
3. Are there any dependencies on proprietary software?
4. Has the product been developed in an open source environment from the outset?
5. Does the product continue to evidence active engagement with the open source community?
6. Is the product's open source development process governed?

[16] Dunbrack Lynne, Holland Marc. RHIO/HIE Market Players Reports. *Health Industry Insights* (IDC). June, 2007.

7. Does the product have a full-time development organization?
8. Is there an adequate menu of whitepapers for the product?
9. Can the product be easily installed?
10. Is the product supported by a full-time support organization?
11. Does the product meet your organization's criteria for functionality, security, and scaling?
12. Can the product meet the core needs of all of the HIE organization stakeholders?
13. Can the product be extended to meet the future needs of your organization?

Managing an Open Source HIE Project

A fundamental question that should be discussed when reviewing the domain of Open Source is how IT project management of an open source implementation would differ from any other vendor-supplied product or internal development activity. Exploring this question can use a well-defined and understood model – the system development life cycle (SDLC) which is used by IT project managers around the world to manage IT acquisition/implementation projects. The SDLC breaks an application development task into several areas, and defines management tasks during each phase of application acquisition/development, testing, implementation, use, and refit/replacement. Whether the project involves use of Open Source code, in-house development, or a vendor-supplied proprietary product, the steps followed, and many of the considerations, are common to all three. Examining the following seven high-level stages in the SDLC reveals how Open Source projects may differ from routine acquisition or development:

- Planning – Define the application boundaries and justify the application, develop an acquisition plan, and develop a process for managing the project to the plan.
- Analysis – Create use cases with the users and IT specialists that illustrate business requirements.
- Design – Develop the logical and technical architecture, infrastructure requirements, user interfaces, and the test cases to support the business requirements.
- Development / Acquisition – Decide on an approach to acquire the application and manage the development process.
    - Open Source projects will appear to have elements from both development and acquisition.
- Testing – Test the developed/acquired application, focusing on the system and user-functional testing.
- Deployment – Place applications into production.
    - Process incorporates development of user guides, training, and implementation management, and often includes post-installation return on investment (ROI) analysis if required by the organization.
- Maintenance – Monitor help desk and support; keep the application current and correct bugs that occur after go-live.
    - Periodic end-of-life analysis, as needed.

Planning

This first step of the SDLC probably has the fewest differences when considering open source. The single most important factor is to educate the organization on the benefits and risks of open source so that the organization can determine if it is willing and able to manage the risks and leverage the benefits. Early in the planning process, once the application has been defined and a rough needs analysis is completed, the project's target success factors (often called "critical success factors") should be determined. The organization's first open source project will require both enlightened departmental management and IT savvy for direction, but there is no reason to be unusually concerned. Good project management skills will suffice to keep the organization on track and provide for a well designed and soundly implemented application, whether open source, acquired, or internally developed.

During the planning process, open source projects can be thought of in the same way as hybrid projects, the only difference is that the initial cost of the open source code may be very low (even zero), but the code extensions and maintenance costs may be greater, depending upon the size of the support vendor and the potential for requested changes to become part of the product's core code base. In most cases, a well designed and managed open source project should have lower overall costs over the project lifetime, but good project management skills are required during all phases to both tightly manage the application evolution and user expectations. It is common during poorly managed projects for user expectations to be misaligned with the delivered product, whether it is open source or proprietary. Open source development organizations that do not spend much on implementation staffing are not strong vendor partners by themselves since they lack established implementation methodologies and strong user-analyst skills. Lack of these skills must be managed, and an experienced project manager will look for outside user-analyst skills when considering an open source development organization that does not have a strong support staff.

During the initial project design it is important to have very open discussions with senior organization management, key departments and respective application user champions regarding operational procedures and process redesign around the envisioned application. This is a recurrent theme throughout this section of the paper since this is the single most important factor impacting a successful installation. "Turn-key" products, which are easily adapted to existing user procedures, may result in less-than-optimal automation in the long run. A good project manager will be vigilant in keeping users engaged and ensuring that process redesign is an integral part of the project.

The functional analysis must match user expectations, and the testing and new operational procedures must be clear and approved by the users. Consequently, senior management must make it clear to those ultimately responsible for the application choice that achievement of a successful install is the responsibility of those who made the application choice.

Analysis

Use case design and development should be independent of the choice of vendor and method of acquiring the desired application. During this phase, projects should pay particular attention to process redesign as mentioned earlier. For those products that have less flexibility during product implementation, the departmental business processes will have to fit more closely, or have to be changed to fit. Alternatively, in an open source environment, new code forks will need to be developed for the project if user processes cannot adapt to the application flow that has been supplied. Aside from a more intensive focus on business process redesign, most other facets of the analysis step should not be different from any other application acquisition or development project.

Design

Upon completion of the analysis phase, the use cases and design requirements should be well documented. Organizations that typically develop their own software will have well developed architectures and processes for application program design. Organizations that typically acquire their applications from vendors have more highly-developed acquisition skills including Request for Information (RFI)/Request for Proposal (RFP) development, product review processes, negotiation, and contracting. Organizations considering open source acquisition should make sure they have a good understanding of their own technical infrastructure, architecture and component services. Vendor-supplied commercial products are often implemented on many different platforms within several different architectures; if an open source product is only offered on open source platforms, the organizations choices may be narrowed.

Development of the RFI/RFP is typically part of the design process where requirements are codified into distinct questions regarding the applicant's processes and functionality. While there are not significant differences on how such acquisition documents are composed, the organization should add sections specifically relevant to open source vendors and solutions, such as those mentioned in the "check list" found in the previous section.

Finally, development of test cases and a test plan is key to making a good vendor selection. Vendors should know what the organization expects relative to application functionality testing, and users, who have provided their input during the analysis phase, should have clear expectations about how they will be evaluating application functionality. Where process redesign has been considered, testing design should be flexible enough to allow vendors to show how their application design will result in user processes that produce efficiencies equivalent to those envisioned during the analysis phase. Open source vendors who have designed their products to address specific user functionality requirements can often have better overall process design, but can also achieve flexibility in user interfaces commensurate with proprietary software vendors. Testing design should focus on user efficiency and not the "lipstick" of flashy UIs.

Development / Acquisition

Both acquisition through commercial vendors and development projects can still have portions of their application architecture satisfied with open source products. Linux is a very popular operating system which is being used by a growing number of vendors of commercial products; it has become a favorite platform for in-house development teams. From a project management perspective, an enterprise's development projects will differ little in their management as a mix of proprietary and open source products are considered and integrated into the developed application.

Acquisition projects will also vary little during the initial steps of creation and release of the RFI/RFP, and during the follow-on paper evaluation steps. Probably most evident will be the "flash" and overwhelming volume of a proprietary vendor's response versus the probably conservative, almost academic, nature of an open source vendor's response. The paper evaluation process should focus on functional capability in the key areas identified by application users, and should pay attention to long-term ROI as part of the evaluation process. In most cases, RFPs for large applications such as full electronic health records (EHRs) will always find proprietary vendors represented in the finalist group when only functionality is considered. A realistic paper evaluation process considers all available information, and will not weight overall initial instillation (or day one) functionality too heavily. When any application is implemented on day one, it is not unusual to find 40% or more of the feature/functions requested in the RFP ending up on the cutting room floor.

Vendor demonstrations are another phase of the RFP/sales cycle, and open source vendors typically will not have the budgets and availability to compete in intensive multi-day product demonstrations. If the organization is serious about evaluating open source products on an *equivalent* basis with proprietary products, then again, the demonstration stage should take key processes and develop the demonstration scenario around those processes. Allowing vendors to demonstrate their solutions to certain scripted scenarios via Web-based meetings will continue to level the playing field since it further reduces the cost to vendors as well as the cost to the organization.

Site references are very important, but typically site visits will yield ambiguous results unless they are carefully managed. Traditional proprietary vendors can swing an application decision through a combination of a willing "demonstration" customer and four-star lunches and dinners. When evaluating open source solutions, good project management will spend the extra time to pose specific scenarios that demonstrate how key functions are performed, and then engage reference customers on Web-based interviews without intervention from the vendor. These interviews should focus on frank discussions not only regarding how the application performs certain functions, but also on the vendor's specific performance at each step of the process, from contracting through testing, implementation support, process redesign, and product evolution.

The decision process should focus on overall ROI and fit of the product to the organization. Contract discussions should begin before the final decision and be well

along when all factors are considered. The ability to work with the vendor during contract negotiations is a key consideration in selecting the right vendor. All costs should be considered in the contract. Proprietary vendors should include all costs for interfaces and functions needed to satisfy the organization's need, including their cost and expectation of any client-incurred costs. Open source solutions, while they may not have a "purchase" cost, will certainly incur many of the same costs as other solutions including interfaces, customization to organizational architecture, and business redesign. In all cases, some of these services may have to be acquired externally. Before a contract is signed, all costs should be considered and documented. Major contract elements must be worked out. Specific attention should address test cases for the application and any contracted code extensions expected to meet. As opposed to proprietary vendors, open source vendors may have no "quarterly quotas" to meet, and consequently may be more willing to wait for all contracting phases to be completed.

Testing

Testing of the chosen product to the scenarios developed during the analysis and design phases needs to take place based on the contracted functions. The user community needs to be *bought-in* to the final decision process, and the features/functions contracted for should be used to customize the testing scenarios. Forcing an application to use test scripts where the acquired functionality is already known to be deficient is both non-productive and harmful to the overall implementation process. Testing should focus on designed outcomes and should not be concerned with how elegant the solution is. In healthcare, elegance often takes a back seat to practicality and desired outcomes.

Since an HIE project is often associated with both acquisition of software and development of functions to integrate the application into the organization's overall technical environment, additional code extensions may also be included in the contract. The timing of delivery and testing of functionality may need to be phased. The project manager must be careful with projects to keep product forks to a minimum, which will hold down the ongoing maintenance cost to the organization.

Upon code turnover, along with the test results there should also be an emphasis on receiving a full set of product and source code documentation. While this should have been examined during the purchase decision, a particular emphasis needs to be placed on the *as-built* documentation since that may be the only artifact that can be used to reconstruct application design when the original supporting vendor is acquired or goes away. It is also important if there is any thought of the organization taking responsibility for its own maintenance.

Deployment

There is virtually no difference in deployment of applications based on their sourcing. If the HIE project has a particular focus on process redesign to meet efficiency objectives, added cost will most likely be required for design of training materials and training.

This is the time when user expectations must be most carefully managed. Since the acquisition cycle may well be separated in time from the actual deployment by many months if not over a year, depending on project size, users will have forgotten what they contracted for, and may have been exposed to more recent and elegant designs since the acquisition occurred. A simple way to keep users engaged is with regular bulletins on testing and discussions/reminders of functional design and process redesign. Working product demonstrations that allow users to see changes in the product are also helpful.

Open source projects will most likely require additional assistance from third parties – experts in implementation of the open source product. In truth, many traditional proprietary vendors rely on third party implementation as well. While an open source product may have been very cost-attractive when evaluating the overall cost/benefit, do not underestimate the inevitable costs for this step. In the latter stages of any application acquisition process, there is typically lack of organizational interest in continued funding for the product installation, and there will be significant pressure on the project to cut costs and get the product live. Such pressure can be singularly disastrous to any product installs if the install cost has been ignored in the initial contracting. To be safe, the project manager should try to get budget funds for installation, training, and process redesign committed and sequestered as soon as possible after the actual contracts are signed (if not in conjunction with the signing).

Maintenance

Open source development organizations usually do not offer end user support services. This means that the HIO must either work with an implementation partner that provides such services or develop those "help desk" services internally. The same choices apply to project-driven application change requests. While an open source development organization may not have organized user groups that collectively drive application evolution, the HIE project management team can either work independently with other organizations using the same product or can work with an implementation/support services partner that has created these user forums. The HIE project management team should review how the chosen product has kept abreast of emerging requirements, such as those being formulated by the Certification Commission for Healthcare Information Technology (CCHIT) (www.cchit.org).

Conclusion

It is clear that the challenges in managing an open source project are virtually identical to managing a project that deploys proprietary software. If, however, the HIO has any ambition to develop proprietary software that is built on open source foundation, then careful attention must be paid to the particular license under which the open source product is released. As noted in the Appendix, the GPL (General Public License) license is not as flexible in this regard as are other licenses, such as L-GPL. If the HIE project management team exercises appropriate due diligence regarding the capabilities of both the open source development organization and the potential open source implementation/support organization (including creating such an organization internally),

and if the team follows through on all of the recommendations outlined above, an open source approach to an HIE project offers no increased risk and also offers some novel factors for risk mitigation.

Appendix

Open Source Software Fact Sheet

Open Source Software (OSS) products are systems whose human-readable ("source") code is always freely available to anyone who is interested in downloading it. This is in contrast to most commercial software, whose source code is considered intellectual property and a trade secret not to be disclosed. Advantages of open source include availability, extensibility, and the opportunity for peer review. Open source products are made available under a variety of licenses, which are discussed below. Although many of the challenges and benefits of using open source software are the same as with commercial software, there are some unique aspects of open source software that need to be kept in mind when selecting a product or vendor.

Questions about Open Source Software

What are the advantages of open source software for healthcare?

Open source solutions have a number of advantages for a healthcare enterprise. The collaborative sharing of ideas and concepts practiced by users of open source software can create 'communities' of developers, partners, testers and users who interact with each other to further improve the software. This can speed up the development process, bringing in skills that a single software vendor would not be able to provide. And the community can also provide an alternative, though unconventional, avenue for technical support.

At the data level, an open source software application does not strand critical health data in a proprietary format. When access to mission critical business data is controlled by an open source application, healthcare organizations are protected from the risk of a technology vendor business failure, or from a merger or acquisition that leads to the sun setting of an installed software solution by the new vendor, and the imposition of a mandatory, expensive and disruptive software "upgrade." Upgrades can be a non-trivial event for an enterprise health care software product. Open Source software increases the bargaining position of a healthcare enterprise, making it possible to "fire" a suboptimal open source software vendor without losing access to the business data.

What are the disadvantages of open source software for healthcare?

Open source solutions present risks for the technology infrastructure of an enterprise lacking prior experience with open source software. Absence of qualified technology staff on site may limit the agility of support for users. In many narrow healthcare verticals there is no compelling open source alternative to the dominant proprietary software vendors. In some cases, there are insufficient options for access to qualified vendor support for the open source solution. There may also be indemnification and liability risks associated with an open source software solution which lacks a well-capitalized vendor to stand behind the product.

Appendix

What is open source software?
"Open source software" refers to the licensing terms governing the use and distribution of the software code as intellectual property. According to the Open Source Definition[1] ten criteria must be met to qualify a software program as "open source":

1. Free redistribution is allowed and royalty payments are prohibited.
2. The program must include source code.
3. Modifications and derived works are allowed.
4. The integrity of the original source code must be preserved.
5. No discrimination against any person or group of persons.
6. No discrimination against fields of endeavor.
7. The license remains with the program even if it is redistributed.
8. The license must not be specific to a product.
9. The license must not restrict other software.
10. The license must be technology neutral.

Open source software has its roots in the 1970s and 1980s when researchers at major universities, such as UC Berkeley and MIT, collaborated to rapidly develop the Unix operating system. During the1990s, when Linus Torvalds launched the Linux kernal project, the use of open source software grew into a mainstream feature of the computer industry. Today, in addition to the open source operating system based on the Linux kernal, there are many enterprise open source software solutions, such as databases (MySQL, PostgreSQL), CRM solutions (SugarCRM), browsers (Firefox), Web servers (Apache), development tools (Eclipse), and more.

What is an open source license?
The legal framework for an open source software license is built on existing copyright and contract law. The original author of the software source code retains the copyright while an open source license is assigned to the software code. Others who want to use the software must abide by the terms of the license. All open source licenses stipulate that the source code is available for inspection and reuse.

What types of open source licenses are available?
Open source software licenses fall into two categories, sometimes referred to as "permissive" and "copyleft." Permissive licenses conform to the ten open source criteria listed above, while copyleft licenses conform to the "Four Freedoms" (see below) published by the Free Software Foundation.[2]

What is "copyleft"?
"Copyleft" (as opposed to "copyright") is based on the Four Freedoms written by Richard Stallman and published by the Free Software Foundation:[2]

1. The freedom to run the program, for any purpose
2. The freedom to study how the program works, and adapt it to your needs
3. The freedom to redistribute copies so you can help your neighbor
4. The freedom to improve the program, and release your improvements to the public so that the whole community benefits

The fourth freedom is also a restriction: by changing the source code, the licensee agrees to release the changed code under the same free software license. In other words, the results must also remain as open source software, allowing the whole community to benefit from all improvements.

What are the differences between "permissive" and "copyleft" open source licenses?
"Permissive" licenses are also called the "BSD style" license, after the original Berkeley Software Distribution license for Unix. The BSD license was based on open collaborative sharing among academic researchers. There are many BSD-style licenses, the most common being the Revised BSD, MIT, L-GPL, Mozilla, Apache and Eclipse licenses. There is only one "copyleft" license, the General Public License (GPL) published by the Free Software Foundation. The main difference between the two types of licenses is that the GPL requires the software and all derivative works to always be licensed under the GPL. In comparison, BSD-style licenses, like academic collaboration, only require acknowledging the original authors, and place few restrictions on derivative uses of the source code.

What is the advantage of BSD-style licenses?
By allowing derivative works to be licensed under a restrictive, non-open source license, BSD style licenses are ideal for mixed enterprise environments, where proprietary software will be integrated with open source software. The absence of the compulsory open source provision for derivative works incentivizes the BSD licensed intellectual property as a library of components that can be easily integrated with proprietary products. BSD style licenses are also used to preemptively release enterprise software solutions under an open source license in an attempt to gain market share advantages. An example of this strategy is IBM's market-making release of the Eclipse SDK environment under a BSD-style license.

What is the advantage of the GPL?
By requiring derivative works to remain under the GPL license, the "copyleft" approach incentivizes the rapid accumulation of a public commons of GPL licensed intellectual property. This programmer-friendly approach is most useful to enterprises that have no need for systemic integration with non-GPL software. The compulsory openness of intellectual property under the GPL can be a powerful competitive advantage for projects with a large community of users and developers. The Linux kernal is licensed under the GPL.

What is the Free Software Foundation?
The Free Software Foundation (FSF) maintains both the GPL license and the Limited GPL (L-GPL) license. The L-GPL is necessary to allow "permissive" style licensing for instances where the GPL is incompatible with existing intellectual property rights.

What is the business model of an open source healthcare software vendor?
Open source software vendors compete for service and support contracts, not for sales. This forces a successful open source vendor to concentrate on customer support, because the absence of proprietary enterprise healthcare software lock-in allows a customer increased opportunity to "fire" the vendor. In addition, the absence of new sales revenue reduces the vendor's opportunity to leverage sales to capitalize new feature development.

If Open Source is such a good thing, why isn't implementation more widespread in healthcare?
There is limited penetration of open source solutions into healthcare enterprises. The absence of development capital for open source projects can result in suboptimal user interfaces and feature

sets among newly developed open source software solutions. On the other hand, general technology adoption in healthcare is at abysmal rates in some markets.

Where can I learn more about open source software in healthcare?
A good place to start is by reading the Linux Medical News Web site.[3] There is a global mailing list named "Openhealth" at Yahoo! Groups.[4] O'Reilly, the technology publisher,[5] specializes in high quality books about open source software. You can also join the HIMSS Open Source Work Group.

References

1. The Open Source Definition is maintained by the Open Source Initiative (http://www.opensource.org), a non-profit corporation dedicated to managing and promoting the Open Source Definition for the good of the community.

2. The GPL and the L-GPL are maintained by the Free Software Foundation. http://www.fsf.org

3. http://www.linuxmednews.com

4. http://tech.groups.yahoo.com/group/openhealth

5. http://oreilly.com

Appendix Fact Sheet is located on the HIMSS website at:
http://www.himss.org/content/files/HIMSSOpenSource.pdf

Credits

Special acknowledgment and appreciation is extended to Tom Jones, Chair of the HIMSS HIE Open Source Task Force, for his time, leadership and content contribution in the development of this white paper.

Members of the HIMSS 2007 – 2008 HIE Open Source Task Force, who spearheaded the development of this white paper, include:

Tom Jones, MD (Chair)
Medical Director
Tolven
tom.jones@tolvenhealth.com

Neil Cowles;
CEO
Tolven
neil.cowles@tolvenhealth.com

Richard Kubica
Managing Director & CTO, IS Infrastructure
Hartford Hospital, Information Services
Rkubica@harthosp.org

Dave Minch
HIPAA/HIE Project Manager
John Muir Health
Dave.Minch@johnmuirhealth.com

Will Ross
Project Manager
Mendocino Informatics
wross@minformatics.com

Pam Matthews, CPHIMSS, FHIMSS
Senior Director, Healthcare Information Systems
HIMSS – Staff Support
pmatthews@himss.org

Holly Gaebel
Coordinator, Healthcare Information Systems
HIMSS – Staff Support
hgaebel@himss.org

The inclusion of an organization name, product or service in this publication should not be construed as a HIMSS endorsement of such organization, product or service, nor is the failure to include an organization name, product or service to be construed as disapproval. The views expressed in this white paper are those of the authors and do not necessarily reflect the views of HIMSS.

