J Pathol Inform

Editor-in-Chief:
Anil V. Parwani , Pittsburgh, PA, USA
Liron Pantanowitz, Pittsburgh, PA, USA

OPEN ACCESS
HTML format

For entire Editorial Board visit : www.jpathinformatics.org/editorialboard.asp

Review Article

Custom software development for use in a clinical laboratory

John H. Sinard, Peter Gershkovich

Department of Pathology, Yale University School of Medicine, New Haven, CT, USA

E-mail: *John H. Sinard - john.sinard@yale.edu
*Corresponding author

Received: 04 September 12 Accepted: 01 October 12 Published: 20 December 12

This article may be cited as:
Sinard JH, Gershkovich P. Custom software development for use in a clinical laboratory. J Pathol Inform 2012;3:44.

Available FREE in open access from: http://www.jpathinformatics.org/text.asp?2012/3/1/44/104906

Abstract

In-house software development for use in a clinical laboratory is a controversial issue. Many of the objections raised are based on outdated software development practices, an exaggeration of the risks involved, and an underestimation of the benefits that can be realized. Buy versus build analyses typically do not consider total costs of ownership, and unfortunately decisions are often made by people who are not directly affected by the workflow obstacles or benefits that result from those decisions. We have been developing custom software for clinical use for over a decade, and this article presents our perspective on this practice. A complete analysis of the decision to develop or purchase must ultimately examine how the end result will mesh with the departmental workflow, and custom-developed solutions typically can have the greater positive impact on efficiency and productivity, substantially altering the decision balance sheet. Involving the end-users in preparation of the functional specifications is crucial to the success of the process. A large development team is not needed, and even a single programmer can develop significant solutions. Many of the risks associated with custom development can be mitigated by a well-structured development process, use of open-source tools, and embracing an agile development philosophy. In-house solutions have the significant advantage of being adaptable to changing departmental needs, contributing to efficient and higher quality patient care.

Key words: Buy versus build, clinical use, custom software, in-house development, open source

Access this article online

Website: www.jpathinformatics.org

DOI: 10.4103/2153-3539.104906

Quick Response Code:

INTRODUCTION

Facing Unmet Needs

Computer software is an integral part of the day-to-day operation of any clinical laboratory. The major nidus for this activity is the laboratory information system (LIS), typically a suite of integrated modules purchased from a single vendor and designed specifically for the operation of the laboratory. LISs have matured substantially over the past few decades, providing greater operational efficiency and improving patient safety.

Yet, even the most advanced LISs do not fully meet the needs of every laboratory. Although some labs may be able to function adequately on their LIS, larger labs and labs providing specialty services typically can identify information management needs which are not met by their LIS. This is because LIS vendors build their software to meet the needs common to the majority of the labs in their current or intended client base rather than to meet the needs of a particular lab, and every lab has some unique needs due to their size, subtleties of their local environment, people, and the clinical focus

of their clients. Thus, a needed functionality is lacking, either because it does not exist at all in the LIS, or because it exists in a way that does not mesh well with the workflow in the lab.

When the LIS functionality falls short of an identified need, labs have a choice: (1) make do with what they have, perhaps adjusting their workflow to accommodate; (2) contract with the LIS vendor to add the needed functionality to their LIS, or to modify it to meet their workflow needs; (3) purchase third-party software which meets the need, (4) develop their own custom software in-house, or (5) purchase a new LIS. Purchasing a new LIS is a huge undertaking and outside of the scope of this discussion. In fact, most labs choose to go with option 1. There can be many ways to accomplish a task in the lab, and labs can adapt to what their LIS is able to do. If the need is great and funds can be identified, option 2 may be chosen. Having your LIS vendor develop integrated customizations assures compatibility with the rest of the LIS, but this can be an expensive and time-consuming process. Nonetheless, LIS vendors rely on at least some clients choosing this option because this funds enhancements to their product. When laboratory science or the regulatory environment creates a general need, the client with the lowest threshold, greatest need, and available capital funds the development of the solution with their vendor. This client has the advantage of dictating how the workflow for the new feature will be designed. After delivery and testing (and payment), the vendor typically incorporates the new feature into a subsequent version of the LIS software, either as a free enhancement or available for an additional charge.

Third-party solutions can be a good option, and certainly should be investigated. If your lab has an unmet need, others probably have a similar need. There are a number of smaller companies that are more nimble than the major LIS vendors and that can respond more quickly to a need and provide a solution. The less specific the need is to pathology (e.g., transcription, image acquisition), the more likely it is that there will be multiple solutions from which to choose. If the solution can operate independent of the LIS, there are no integration concerns. If it needs to be integrated, the company may be able to handle it themselves, unless modifications are needed to the LIS system, in which case the third-party company will then likely have to enter into some sort of agreement with the LIS vendor. There are many examples of successful third-party solutions, the most common of which is "middleware" in the clinical laboratories, which manages communication between analytical instruments and the LIS.

However, if the need is novel or specific for your environment, third-party products that adequately meet that need are often simply not available. In this situation, in-house custom software development may be the best solution.

OUR EXPERIENCE

At our institution, the pathology department provides anatomic pathology services only (laboratory medicine is a separate department). We have our own Pathology Informatics Unit that provides both operational services (i.e., information technology services) and software development. We also have three other full-time faculty members with academic informatics programs, but they are not involved in the software development for clinical use. Our clinical development team has developed a variety of clinical applications that are used every day in our anatomic pathology practice. Some of these solutions are integrated into and/or interact with our LIS, and some are standalone solutions. Our development team consists of three people. One is a pathologist, who also has other significant clinical and administrative needs and thus spends only about 20% of his time on software development. His primary role is in specification development and in programming the department's LIS to interact with the custom software (we have a unique arrangement without LIS vendor which allows us to introduce our own customizations into the commercial software, which was nicely designed to allow for this process). Another developer handles the user interface creation (according to the provided specifications) and also handles deployment, user training, and initial support of the custom applications. The third developer spends approximately two-thirds of his time on development, predominantly the business logic of the standalone components of the application, with the other third spent on management of the operational informatics unit in the department. Thus, in total, this represents about 1.6 FTEs (full-time equivalents) of true development resources. Our LIS is Cerner CoPathPlus. Our standalone components are developed in Java, predominantly as web applications.

Over the past several years, solutions we have developed and deployed include: Digital image file management,[1] scanned document file management,[2] dictation/transcription management,[3] an outreach support system for orders and report delivery, an outreach client interface system, a repetitive task scheduling engine,[4] frozen section management and diagnosis communication to operating rooms,[5] histology asset tracking,[6,7] trainee diagnosis tracking and evaluation,[8] and numerous databases to support trainee interviewing and recruitment, graduate trainee tracking, computer hardware tracking, research histology, and graphics services billing. This article is based on our collective experience with this development and deployment process.

BUY VERSUS BUILD

Medical institutions operate around a number of philosophies, some codified in actual policies, but most driven by the experiences and/or preferences of the

institutional leadership. In the information technology (IT) departments, one of the most prevalent dichotomies is the preference to buy or build: When an unmet IT need arises, does one buy a commercially available solution or invest in building/developing a custom solution? Although there is a lot of software commercially available, it is generally not designed for some of the specialty medical uses and may not mesh well with the existing workflow. In many environments, the task of assessing the suitability of available software to meet a specific need is often relegated to IT staff who are not that familiar with the clinical workflow. A solution is purchased and installed which does not fit quite right. Since the "off-the-shelf" software is generally not modifiable by the end-user, departments then find themselves adjusting their workflow to match what the software can do. Advocates for the "build" philosophy argue that computers are a tool that should be adapted to the user's workflow rather than dictating a particular workflow to its users.

Advocates for the "buy" philosophy raise a number of common objections to development of software in-house. These, along with counter arguments, are listed in Table 1. In addressing any unmet software need, it is always prudent to explore what solutions might be commercially available, either through your LIS vendor or from a third-party. There is a lot of software available commercially from a large number of vendors. How "thorough" of an investigation one does depends, of course, on the need and potential benefits, since exploring options can be time-consuming. If your need is unique, however, there may simply be nothing available which comes close. We experienced that when we developed the specifications for our Frozen Section Management and Diagnosis Communication software.[5] The buy versus build cost/benefit analysis needs to consider the true cost of ownership, not simply the difference between the purchase cost and the development cost. On the buy side, one must include the costs of installing and setting up the new software, included in the cost of most large vendor offerings, but sometimes an additional charge from smaller vendors. Are there additional hardware costs associated with the new software, like workstation upgrades? Are there expenses associated with integrating the new software into your workflow, including any workflow changes and perhaps personnel changes? Does the solution require purchasing specific consumables such as a particular vendor's labels, cassettes, or slides? There may also be subsequent annual support costs, which often run 22-25% of the initial purchase price. Collaborating with your LIS vendor to develop the solution as part of the LIS is an option, but can be expensive, and there will almost certainly be annual support cost increases. Additionally, the vendor will be limited by the technology used to develop the LIS, often technology that is a decade old. Determining the cost of developing software in-house can be difficult, but it usually comes down to people and time—the faster you need the software completed, the more people it may take. The more people you hire, the more expensive it is, and then there is the issue of what are you going to do with those people after the software has been developed? (This is discussed further later)

On the other side of the balance sheet, determining a "return on investment" for any software implementation is very difficult because the software typically does not generate new income but rather improves the operational efficiency of the entire clinical service. Even if one could objectively measure staff efficiency, productivity, or frustration levels, there are many other variables that contribute those practice characteristics. However, the greatest benefits to staff efficiency are obtained when there is a very high compatibility between the software and your "ideal" workflow. For third-party solutions, how good is the fit between what the software was designed to do and what you need it to do? For solutions developed in collaboration with your LIS vendor, how

Table 1: Buy versus build: Point/counterpoint

Advocates of "buy"	Advocates of "build"
Anything you might want, someone has already written something pretty close	Do you adapt your workflow to match the software or design the software to match your workflow?
Anything we develop will not be as good as what our vendor can do for us	Vendor customizations are expensive and take time; they are constrained by the capabilities of the tools they chose to develop their product; having mature products, they are often less inclined to innovation
Custom development is too expensive	Increased efficiency and productivity from software designed for your specific environment can save money; need to consider total cost of ownership
Software development requires too many people	Software development requires the right person/people; can be done with one person
We are not in the business of software development	We are in the business of technology development and adoption
What if the person who develops it leaves after a few years?	(a) You got a few good years out of it (b) If it fails the day they leave, it was probably too expensive to maintain anyway (c) If it is useful and saves money, hire someone else to maintain it and/or rewrite it using even newer technology

well can you articulate your needs in the form of specifications? The primary advantage of custom software development is that it is likely to result in the greatest efficiency improvements because it can be specifically tailored for your unique environment, which may include consideration of issues such as preferred and supported platforms, level of IT support available, other institutional systems and policies, idiosyncrasies of the users, and specific elements of the workflow. Additionally, unlike most commercially acquired software, software developed in-house can be adapted and modified as the needs become better defined through use and/or change over time. In other words, your custom solution will become an up-to-date reflection of your operational strategy. You will also likely find that once a solution to a problem has been developed and deployed, other related problems can be addressed by adding incremental functionality to that solution. For example, in our environment, we discovered that our in-house developed solution for digital image file management could be extended to semi-automate the management of scanned documents. This answers the question about what your developers are going to do once they have developed the solution you hired them to develop.

One of the most common arguments against software development is the feeling that it requires too many people to do it right. This is not true. Significant improvements in workflow can be obtained with even just one person. Ultimately, it is an issue of scope and time—how large is the project and how quickly does it need to be done? Remember that very little even custom software is written from scratch any more. There are a variety of software frameworks, libraries, and other components available, many for free, some for a small fee, which perform a wide variety of tasks, and most of what the software developer is doing is building a wrapper around these components which ties them together into an integrated system which meshes with the workflow in the department. Software development is not so much about finding enough people but rather about finding the right people with complementary skills that interact in a synergistic fashion. While it is true that having a large number of developers increases the chances that some of them will be the right people, in the end you only really need the right ones. Finding the right people, however, is not easy. Several studies have shown that the individual productivity of software developers with comparable levels of experience can vary by a factor of 10.[9] Many individuals who market themselves as "programmers" are really more super-users than programmers. Be sure to require any applicants to submit examples of things they have written, and do a quick web-check to be sure it is not simply something they downloaded from someone else. Ultimately, however, their performance in your environment and in the pathology domain will be the key determinant, but it often takes longer than the standard probationary period to determine whether or not a particular person is right for the team. Over the years, our team has had five other individuals who were either full-time or part-time "programmers" before settling on the current team of three. Shortcoming of those no longer on the team included lack of problem-solving skills, poor attention to detail, and in more than one case simply an inability to understand and adapt to the clinical workflow of an academic pathology department.

Many laboratories will claim that they do not engage in custom software development because they are "not in the business of writing software." However, every high-end clinical laboratory, especially those at "cutting edge" academic medical centers, explores new testing technology when it becomes available. When what is commercially available lags behind new scientific developments, many of these labs will either collaborate with vendors to develop the needed technology or develop and validate it themselves as a "laboratory-developed test." This is because pathology laboratories are in the business of technology adoption and development in order to make the most modern diagnostic testing available to enhance patient care. Software is simply another technology—a part of the laboratory's strategy to provide better clinical services – and as such falls well within the scope of "the business" of laboratories.

Finally, one of the most common arguments made against developing clinical software in-house is the concern over who will support the software if the person who developed it leaves the institution. Clinical labs have a responsibility not only to their financial health but also to the patients on whose specimens they perform testing, and the clinicians who care for those patients. To assure longer term stability, most healthcare IT directors prefer to restrict the source of software for clinical purposes to large commercial vendors who have been around for years and are likely to continue to be around for additional years. Smaller third-party vendors offering acceptable solutions raise concern about the company's stability, especially in the current environment of numerous technology company failures. Custom software development is often discouraged. In fact, for a number of years after the creation of the informatics unit at our institution, we resisted going down the path of custom software development precisely to preserve external supportability. However, after many years of struggling with workflow inefficiencies caused by insufficient software, we ultimately decided that even if we only got a few years of use out of a custom solution, the efficiency gains during those few years would be sufficient to justify the development costs. Our vulnerability to the possible unexpected departure of a custom solution developer has been lessened by the extensive use of open-source software (OSS) (see discussion below) and by developing

our software using a team approach, as it is unlikely that the entire team would leave simultaneously. In reality, even if a key developer were to leave, the existing software does not stop functioning. If it does, that would suggest it required constant maintenance, and then it was probably too expensive to maintain anyway. If it is a very valuable piece of software, a "quick repair" could be subcontracted to get the solution working again while other or new developers are hired to either support the existing software or redevelop it using newer development tools, likely resulting in a superior end result.

CUSTOM SOFTWARE DEVELOPMENT PROCESS

There is no one right way to develop custom software, but there are some standard processes that tend to yield superior results. The step-by-step process is summarized in Table 2.

Functional Specification

The most important part of custom software development is preparation of the specifications document(s). These are often divided into functional specs and technical specs, but ideally can be combined into a single document depending on the scope of the project and the breath of knowledge of the individuals involved. If done sequentially, the functional specification has to come first, and is the more important of the two. If the functional specifications are not complete, thorough, and well conceived, a lot of wasted effort and re-engineering will be required later in the development process. The functional specs describe the look and feel of the software to be developed. It requires a clear understanding of what problem the software is supposed to solve, and how the solution will be used in the workflow. It is best if this document is prepared by a pathologist rather than by an "IT" person. It should not be a "high-level" document, but rather should be very specific, ideally down to diagrams of the screens and a description of what should happen when the user interacts with each control (button, checkbox, drop-down list, etc.) on the screen. End-users may not be able to conceptualize the application at this level. Rather, they will provide "use-cases", descriptions of when they would use the software and what they need it to do. The author of the functional specification then needs to take these use-cases into account, assimilate the information, and design an application that meets user's needs. The design should take into account where the software will be used (screen size, processor and memory requirements, space for other resources such as keyboard, mouse, and scanners). Finally, appropriate consideration needs to be given to whom the primary users are going to be, and that includes their likely skill set and personalities. What are the users willing to do? For example, if the designed software requires too many mouse clicks, too much typing, or has delays that routinely exceed half a second, many pathologists will be very resistant to adoption, and the solution may fail simply because users avoid using it. It is very useful if multiple potential users can review and have input on the functional specification, both to help assure the solution is general enough to be usable across multiple subspecialties (if appropriate) and to obtain greater buy-in from the user base.

Table 2: Custom software development process

Specification development – Functional specification
Based on use-cases
Needs – What does it need to accomplish?
Workflow integration – How is it going to be used?
Environmental integration – Where is it going to be used?
Personnel integration – Who is going to use it and what are they likely to do?
Specification development – Technical specification
Storage – What will the database structures look like?
Software architecture – How will the software be structured and delivered to users?
Scalability – How will it hold up to increasing use and users?
Maintenance – How much maintenance will it require and who is going to do that?
Software development
Tools – What programming tools will be used to build it? Open-source options
Test environment – Need a development environment to protect production system
Validation – Assuring that the software performs as expected
Documentation – What documentation requirements are there?
Deployment
Transfer to production – Can the deployed software be kept isolated?
Pilot phase – Selection and training of initial users
Fine tuning – Get feedback from users and adjust software as needed
Back-out plan – Can you go back to where you were if it does not work?
General training and deployment
Post-deployment assessment
Updates and enhancements
Bug detection and correction
Future enhancements
"Scope-creep" and expansion to related problems

Technical Specification

When the functional specs are completed, the document can be handed-off to IT staff to develop the technical specification. Depending upon the scope of the project, multiple technical individuals with different areas of expertise may be appropriate (e.g., application developer and database designer). During this phase, the data structures will be developed and the overall architecture of the application will be designed. The most appropriate

application delivery method (e.g., desktop application, client-server, web application) needs to be determined since this will affect the structure of the application. Typically, splitting the solution into modules is desirable, especially if the modules are independent enough that they can be developed and deployed gradually. The technical design phase also has to take into account the scalability of the application. How many total and/or simultaneous users will there be? How much data will the application accumulate? These issues have implications for database table structure, indexing, memory requirements, etc. Finally, it might be a good idea, even at this phase, to consider the maintenance needs of the application. How much ongoing maintenance will be needed, and who will do it? For example, a solution that automatically files a high volume of material but requires manual intervention to process failed filing events will need ongoing resources for the manual processing.

In our environment, the majority of the specifications are combined, functional and technical, and are prepared by a pathologist with informatics experience. Specification documents can vary in length significantly based on the scope of the application, and ours have ranged from about five pages to over 100. The specification document(s), particularly the technical components, should be considered a dynamic document, and may have to be modified as the application is developed to fine-tune the final product.

Software Development

The actual software writing is the most variable part of the development process. Individual programmers will have different skills, and different preferences for development platforms (operating system as well as integrated development environments) and programming languages. The use and integration of open-source frameworks and components can substantially facilitate this process, as well as mitigate many of the risks often associated with custom software development. OSS is software developed by self-selected groups of programmers and made available to the public for use, modification, and incorporation into their own solutions. Many OSS projects have some management infrastructure and follow formal process (e.g., Free Software Foundation, Apache Foundation, Mozilla Foundation, etc.). Typically, this software is available for free, but may carry some restrictions about future commercialization, which is usually not a problem for pathology departments developing custom solutions to problems because there is no intention to subsequently commercialize the product. There are thousands of OSS projects available for download at GNU Savannah (savannah.gnu.org), sourceforge.net, Google code, etc. Well-written solutions are very popular, and as such are extremely well tested and debugged by multiple users, and updates with new features and some bug fixes are routinely produced. OSS can include fully operational solutions (e.g., Linux operating system, MySQL database, Apache Web Server, Apache Tomcat), development tools (e.g., Eclipse integrated development environment, Google Web Toolkit), as well as frameworks (Sprint, Hybernate, Quartz, etc.), component libraries and application programming interfaces (e.g., HAPI application programming interface, iText, jFreeChart, Apache Commons) that can be integrated into custom software such as HL7 engines, pdf generation, scheduling engines, and charting/graphing solutions. Use of open-source components substantially mitigates the risks of custom software development because it markedly reduces the volume of code to be written, expedites development, and minimizes bugs and errors because significant portions of the solution have already been tested by a large number of users. You retain the ability to modify the software to meet your current or future needs.

There is substantial literature on Rapid Application Development, and various methodologies have been used to improve productivity and deliver working code to the end-user in the shortest amount of time, maintaining a balance between development time and the quality of the end product. The essence of these approaches is outlined in the Agile Software Development Manifesto.[10,11] It was critical for us to adopt these Agile Principles in our development process because the speed of implementation of new functionality is one of the key benefits of in-house code development. Open source, with its philosophy of "release early, release often," feeds well into this methodology.[12] This philosophy is typically well received in medical environments because users can see ongoing improvements and more importantly can provide feedback and input which may alter the direction of the project toward a superior final product.

Finally, developing custom software for clinical use obviously requires a "test environment." The clinical mission of the department must go on, and cannot be compromised by the occasional runaway process and/or system crash that invariably occurs during software development. As much as possible, the test environment should mimic the production environment.

Validation

Custom software for clinical use should be treated like any other laboratory-developed test and appropriately validated before it is used in a clinical setting. Validation (not to be confused with autovalidation of test results) may be governed by regulatory statutes, and those need to be considered carefully (Note that while we hope the discussion here proves helpful, it should not in any way be considered a comprehensive evaluation of the current legal and regulatory environment surrounding software development for clinical use). The Food and Drug Administration (FDA) of the United States Federal Government has oversight responsibility and authority

for "medical devices." Software integrated into medical devices has always, therefore, been regulated by the FDA. The FDA does consider LISs in general to be medical devices, but these have been exempt from the 510(k) approval process since June of 1988.[13] In contrast, because of its charge to regulate blood products, the FDA in March of 1994 determined that any software used in healthcare "for the maintenance of data that personnel use in making decisions regarding the suitability of donors and the release of blood or blood components for transfusion" is required to undergo the 510(k) approval process.[14] Given this requirement, development of any custom software that touches on blood product management should be approached cautiously. It is worth noting that in 2011, the FDA issued a new regulation pertaining to what they define as "Medical Device Data Systems" (MDDSs).[15] This includes standalone software which stores and/or displays data derived from medical devices. This new regulation reclassifies MDDSs from Class III devices to Class I devices, substantially softening the regulatory requirements for bringing these systems to market. Developers of such software, including healthcare facilities, are required to register and list with the FDA. With respect to custom software development for in-house use only, however, the Code of Federal Regulation which governs the operation of the FDA specifically defines as "exempt from registration" any "licensed practitioners, including physicians, dentists, and optometrists, who manufacture or otherwise alter devices solely for use in their practice" (21 CFR 807.65(d)).[16] Thus, there does not appear to be any current requirement to notify the FDA if you are developing software for use within your own institution, but this does not exempt the developers from validating their software. Moreover, as laboratorians, pathologists recognize the value and need for good quality control practices in the generation and distribution of any data used for patient care. The FDA does provide some useful guidance in this regard. In their Laboratory Manual of Quality Policies under Test Methods and Validation, they state "5.4.7.2 Computer Use: When computers or automated equipment are used for the acquisition, processing, recording, reporting, storage or retrieval of test data, if computer software is developed by the user, its development is documented in detail and algorithms are validated."[17] Additionally, in 2002, the FDA released a guidance document entitled "General Principles of Software Validation."[18] This document defines software validation as: "Confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled (Section 3.1.2)." The guidance document acknowledges that determining how much testing is "enough" is difficult and that a developer "cannot test forever." Rather, the goal is to achieve a "level of confidence that the [software] meets all requirements and user expectations."

From a very practical standpoint, the extent of the validation needed depends on what the software does. Does it create new data, or does it simply display data already captured/created from other validated systems in a new way? Will it be immediately obvious to the user if the software is not functioning properly, or is there a risk that using the software could result in an inappropriate clinical action? The greater the risk, the more extensive the validation activities need to be. For low-risk solutions, end-users can be part of the validation process via pilot phases. Users typically will not "accept" software that does not do what it is supposed to do, especially if they know that that software was developed in-house and thus could be fixed. Finally, an important part of the validation process is documentation. This includes the initial design specifications, documentation within the code itself, documentation of testing, and detailed instructions of what changes are needed to the production environment for deployment.

Solution Deployment

Deployment strategies for custom software can vary significantly and are dependent upon the scope of the solution developed. Is the code a standalone solution, an integrated solution, or are there components of both? What kind of concurrency is required between the two? For example, if you have engineered a composite solution with some standalone software and some software integrated into your LIS, can the standalone component exist without the presence of the LIS integrated component, or do they both have to be deployed simultaneously? Once deployed, is use of the software optional (e.g., an alternate way of doing something) or is it integral (it now becomes the only way to do something)? Either way, the code ultimately needs to be transferred to and set up in the production environment, and this is a great opportunity to double-check the documentation you have developed to make sure it is complete. The documentation should include all the components of the solution and any changes needed in the production environment for the software to operate properly.

If the software can be kept somewhat isolated (i.e., its use is not required in the workflow), a pilot phase is strongly recommended. In this phase of the process, a select group of users is chosen to try out the software for some or all of their cases for a period of time. Whenever possible, this should include those pathologists involved in the functional design of the software. Important feedback can be obtained from using the software "for real," and this may result in some modifications or update to improve the synergy with the workflow.

If the software cannot be kept isolated (i.e. its use, upon deployment, becomes required), then a back-out strategy

needs to be developed in advance. What will you do if the software does not work as expected? It is usually not feasible to simply stop all work in the department while it is being fixed. There are times when deployment has no practical back-out strategy (such as upgrading the LIS to a new version), and in such instances it is typically prudent to perform the deployment on a weekend when there is less time pressure associated with returning the system to a fully operational status.

For large development projects with multiple modules, one may have to choose between deploying modules individually or all at once. A staged deployment is often preferable, since users seem to respond better to smaller, incremental changes than to complete workflow changes. One may also have to decide between applying a workflow change to a subset of the specimens initially, or to all of the specimens at the same time. While the former may seem to carry less risk, remember that you are then creating two different workflows (the new and the old) which are operating side-by-side in the same lab, and that can be more disruptive than applying the change to all specimens from the start.

User training is always advisable in advance of any deployment, although this can be harder to accomplish in practice than in theory. Users do not typically have a lot of spare time in their workday, and there can be little incentive to learning something new that is not immediately applicable. We have found it more effective to simply inform everyone well in advance (about a week), give the new users an overview of what the new workflow will look like, perhaps allow them to choose when would be the best day/time for the deployment, and then provide significant on-site support at the time of the roll-out, with the developers and ideally the pilot users and functional designers present/available to help answer questions (which often take the form of "why do I have to do it this way?" rather than "what am I supposed to do next?").

Careful monitoring of the performance of the new software is needed following deployment. Despite extensive testing, there is no way to adequately anticipate the variety of ways users will find to use the new software, and this may uncover subtle deficiencies in either the design or the development which have to be addressed. Remember that many users are pretty clever and may find ways around using the software as intended, especially if it sometimes does not work quite right. Users may identify problems, but simply work around them rather than notifying anyone. We have found it useful to remind users about a week after deployment that the development team needs to be informed of any unusual behavior or possible bugs. It is not an infrequent occurrence at our institution where, for example, a pathologist involved in the development notices weeks later when they are on service an issue with the software which, when queried, everyone acknowledges having noticed but no one made any attempt to notify the development team about.

Updates and Enhancements

Invariably, small bugs will be identified in any custom software which need to be addressed, but the frequency of detection does drop-off quickly after the software has been used for a couple of weeks. Sometimes, developing and deploying what is thought to be the ideal solution provide a better understanding of the problem, and that better understanding may suggest modifications to the software to create a superior solution. More commonly, an onslaught of suggestions for additional improvement will arise, and this should be interpreted as a sign of great success. Having seen what it is possible to accomplish with custom software and the workflow improvements that can result, users are inspired to think of additional improvements that will provide even greater benefits. It is important to have a process for aggregating these suggestions, appropriately vetting them with the design team. Is this a general improvement, or something that satisfies the idiosyncrasies of an individual user? Does it move the solution in the desired direction, or backward closer to the original workflow? Is it a small change or a large one that will require substantial re-engineering? It is often valuable to allow the deployed software to incubate with users for a while before rapidly responding to requests for changes. However, once the developed software has stabilized in the workflow, new opportunities for even greater workflow enhancement can be explored. Once a solution has been developed, incremental additions can progressively expand the scope of the solution to solve additional, related problems. This flexibility and the capability to respond quickly as needs become better-understood or change, or as additional needs arise, without having to embark on a new series of vendor negotiations is one of the true values of custom software development.

From the developer perspective, there are two relatively common pitfalls to be avoided, and these depend on the personalities of the programmers. The first is to abandon ownership of the application too early. Developers tend to like to develop, not support. But support groups within the department or institution will not understand the new application. They will not know what it is supposed to do, and what it can do. Integral involvement of at least a portion of the development team with the initial users can be critical to proper use and acceptance of the new software. Additionally, if any unanticipated behavior of the new solution is uncovered, it is an opportunity for the developers to see that first hand within the real-life use of the application. The second pitfall which developers can fall into is to never abandon their application. Developers can become attached to the product of

their efforts, and may take to repeated reworking and refactoring of their solution to make it a little better. Perhaps a little redesign of a particular routine will make it run a little faster, or make it a little more generalizable. These activities, which typically affect the "back-end" of the application but have no visible effect to the users or the functionality, need to be examined critically with respect to what value they are truly providing. Avoid the trap of updating the application each time a new version of a library or other tool becomes available because this can consume significant resources with minimal yield. Even after a number of years, some developers may wish to cling to their initial code beyond its practical viability. Five or so years after the initial writing, it may be faster, cheaper, and more reliable to rewrite the solution from scratch using new tools rather than trying to update prior code. The logic for the solution has already been developed and worked out and usually can be transferred directly to a new development environment with minimal modifications.

Project Leadership

One aspect of software development that has been absent from this discussion is that of project leadership. This is largely because the software development team in our Pathology Informatics Unit is rather small (essentially three people) and the decision-making hierarchy is clear. For larger groups, however, it is important to have a single individual ultimately responsible for the application. This individual should approve any changes to the specifications during the development process. They need to understand all facets of the solution and the workflow in which it will be used, and have the clarity of thought to be able to anticipate the implications for how modifications to one part of the application will affect other parts.

CONCLUSION

Despite commonly voiced and exaggerated concerns over the costs and risks associated with custom software development, tremendous yields in productivity and efficiency can be achieved with relatively modest investments. Both the costs and the risks can be mitigated by incorporating OSS into the solution, and by having a well-structured development process. Having the right people involved is far more important than the number of people, and involvement of individuals with a deep knowledge of the workflows which may be affected is crucial to the success of the development process. Ultimately, software is a tool. That tool needs to fit your needs, your environment, and your workflow. That tool needs to be adaptable in a time frame consistent with the changing practice of clinical laboratories, and has to strategically advance the mission of the lab to provide the highest quality patient care. The focus should be not on how the tool comes into existence, but on the benefits obtained from its use. When commercially available solutions fall short of the needs, custom software development is a viable, often superior solution.

REFERENCES

1. Sinard JH, Mattie ME. Overcoming the limitations of integrated clinical digital imaging solutions. Arch Pathol Lab Med 2005;129:1118-26.
2. Sinard JH, Gershkovich P. Semi-automated archiving of scanned requisition documents in anatomic pathology. [Abstract: Advancing Practice, Instruction and Innovation through Informatics, 13th Annual Meeting, Pittsburgh, PA] Arch Pathol Lab Med 2009:133:1155.
3. Gershkovich P, Sinard J. Integrating digital dictation and anatomic pathology LIS. [Abstract: Advancing Practice, Instruction and Innovation through Informatics, 14th Annual Meeting, Pittsburgh, PA] Arch Pathol Lab Med 2010;134:938.
4. Sinard J, Gershkovich P, Freis W, Daley A. Use of a repetitive task scheduling engine for workflow automation and rare event detection in a clinical environment. [Abstract: Advancing Practice, Instruction and Innovation through Informatics, 13th Annual Meeting, Pittsburgh, PA] Arch Pathol Lab Med 2009;133:1155.
5. Gershkovich P, Mutnick N, Sinard JH. FSLink – Frozen section management software with real-time communication with the OR. [Abstract: Association of Pathology Informatics, 2010 Annual Meeting, Boston, MA] J Pathol Inform 2010;1:27-8.
6. Sinard JH, Mutnick N, Gershkovich P. Histology asset tracking: Hidden practices and their consequences. [Abstract: Association of Pathology Informatics, 2010 Annual Meeting, Boston, MA] J Pathol Inform 2010;1:4-5.
7. Sinard JH, Mutnick N, Gershkovich P. Histology asset tracking dashboard: Real-time monitoring and dynamic work lists. [Abstract: Association of Pathology Informatics, 2010 Annual Meeting, Boston, MA] J Pathol Inform 2010;1:6-7.
8. Sinard JH, Mutnick N, Gershkovich P. Facilitating feedback and education on the hot seat rotation. [Abstract: Pathology Informatics, 2011 Annual Meeting, Pittsburgh, PA]. J Pathol Inform 2011;2:43.
9. Mills, HD. Software productivity. Boston, MA: Little, Brown; 1983.
10. Beck K, Beedle M, Bennekum A, Cockburn A, Cunningham W, Fowler M, *et al*. Manifesto for agile software development. Available from: http://www.agilemanifesto.org/. [Last accessed on 2012 Sep 1].
11. Beck K. Test-driven development by example. Boston, MA: Pearson Education Inc; 2003.
12. Raymond, ES. The cathedral and the bazaar. Available from: http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html. [Last accessed on 2012 Aug 18].
13. US Department of Health and Human Services Food and Drug Administration Center for Devices and Radiological Health. "Guidance for Industry, FDA Reviewers, and Compliance on Off-The-Shelf Software Use in Medical Devices"; September 9, 1999. p. 16. Available from: http://www.fda.gov/medicaldevices/deviceregulationandguidance/guidancedocuments/ucm073778.htm. [Last accessed on 2012 Sep 1].
14. US Department of Health and Human Services Food and Drug Administration Center for Biologics Evaluation and Research. Software used in blood establishments. Available from: http://www.fda.gov/downloads/Biologics Blood Vaccines/Guidance Compliance Regulatory Information/Other Recommendations for Manufacturers/Memorandum to Blood Establishments/UCM062804.pdf. [Last accessed on 2012 Aug 18].
15. US Department of Health and Human Services Food and Drug Administration Center for Devices and Radiological Health. Medical Devices: MDDS Rule. Available from: http://www.fda.gov/Medical Devices/Productsand Medical Procedures/General Hospital Devices and Supplies/Medical Device Data Systems/ucm251897.htm. [Last accessed on 2012 Aug 18].
16. United States Code of Federal Regulations. Title 21. Part 807. Section 65. Available from: http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm?fr=807.65. [Last accessed on 2012 Sept 1].
17. US Department of Health and Human Services Food and Drug Administration Office of Regualtory Affairs, Division of Field Science. Vol. I. Section 5.4.7.2:

Laboratory Manual of Quality Policies for ORA Regulatory Laboratories; 2012. p. 33. Available from: http://www.fda.gov/Science Research/Field Science/Laboratory Manual/default.htm. [Last accessed on 2012 Sep 1].
18. US Department of Health and Human Services Food and Drug Administration Center for Devices and Radiological Health. : General Principles of Software Validation; Final Guidance for Industry and FDA Staff; 2002. p. 6. Available from: http://www.fda.gov/Medical Devices/Device Regulationand Guidance/Guidance Documents/ucm085281.htm. [Last accessed on 2012 Sep 1].