



Factors to consider before building your own DAM

Cheryl Scheer

is the Multimedia Archivist for August Home Publishing, a publishing/media company in Des Moines, Iowa. She is establishing archiving processes to facilitate repurposing of digital resources. She is part of the team developing a digital asset management system for the company.

Keywords: *digital asset management, DAM, open source, Linux, corporate culture, software development, buy or build*

Abstract Despite the availability of commercial DAM systems, some organizations — including the author’s employer, August Home Publishing — choose to build solutions using in-house or contracted expertise. This paper explores important factors to consider in a “buy vs build” discussion. It will also discuss some open source technology options that can be utilized to develop a digital photo archive solution and an overview of the expertise that will be required.

Journal of Digital Asset Management (2007) 3, 17–22. doi:10.1057/palgrave.dam.3650057

BUY OR BUILD — WHY THE DEBATE CONTINUES

Enterprise DAM software solutions have been commercially available since 1990s. The metadata management concepts implemented in these systems are far older, reaching back to long-established library methods for describing and organizing information. Most of these solutions utilize standard databases to store asset metadata. Considering this history and the existence of proven commercial solutions, why does the “buy vs build” debate still occur with such regularity?

Organizations that decide to build their own business solutions often cite two primary reasons: cost savings and customized functionality. This rationale is valid but incomplete, even if the organization already has the necessary technology resources and human expertise available. There are many other factors to consider before taking the leap into software development. Every organization must review all options and review available commercial products and decide on a solution that best fits its needs. For most organizations the best decision will be to purchase a solution, but others will find that a homegrown solution will be feasible if not preferable.

An in-house software development project will be challenging but can be successful if the organization is committed to all facets of development and long-term maintenance. This

commitment may entail a significant change in the organization’s focus. It is not enough that a small core group has the desire and skills to build a solution. Programming expertise aside, the prevailing culture of an organization can have a significant impact on a project’s chance for success and ongoing survival and this will also be explored in more depth later in this paper.

PERCEIVED ADVANTAGE OF BUILDING: COST SAVINGS

Most DAM vendors will hedge if you ask how much their product costs, and rightly so. Every customer and every installation has unique requirements so it is difficult to quote an “out-of-the-box” price. That being said, often the entry price for an enterprise solution will run into the tens of thousands of dollars, with some systems starting at over \$200,000. These players are very well known — they offer powerful, effective solutions and long lists of customer references.

Many smaller companies and nonprofit agencies have functional requirements that rival those of a large, wealthy corporation but they lack the financial resources to make the initial cash investment. A company may not be willing or able to risk a large expenditure without absolute assurances of a return on that investment. Organizations often find that a few

Cheryl Scheer
August Home Publishing,
2200 Grand Avenue,
Des Moines,
IA 50312, USA
Tel: +1 515 875 7023
E-mail: cscheer@
augusthome.com+DB

of their key requirements are only addressed by high-end solutions. They cannot justify purchasing unneeded functionality without a solid plan to expand into it.

Several DAM solution vendors now offer tailored solutions to meet the needs of these potential customers. Flexible licensing, hosted solutions and upgrade paths offer more options to customers with smaller budgets. Nonetheless it remains difficult to illustrate the ROI that could be gained by implementing a DAM system.

Even those who fall in the “buy” camp can be faced with the stark reality that the only way to get a DAM solution is to build one. Over the past few years open source tools — databases, programming languages, software modules, etc — have gained greater mainstream acceptance and have, no pun intended, opened up new avenues for software development.

The growth and stability of open source development tools has led to three options that put DAM solutions within the reach of more organizations:

- Commercial solutions built with open source tools and/or databases and offered at more affordable prices.
- Complete solutions made available under open source licenses and customizable to meet specialized requirements.
- Development tools and software modules for organizations who wish to build their own solutions.

The cost savings gained through using open source tools has been both celebrated and maligned. Later in this paper we will further explore this issue in a little more detail.

PERCEIVED ADVANTAGE OF BUILDING: CUSTOMIZATION

Virtually every off-the-shelf DAM solution can be (and often is) customized to address unique customer requirements. If a commercial solution requires customization, the wisest option is to either pay the vendor to do it, or contract with one of the vendor’s development partners. This helps to ensure that the solution meets requirements but it can significantly increase costs. A major advantage to taking this route is that the vendor or contractor is generally

responsible for maintaining and updating the custom code over time.

Another reason often cited to support the decision to build is that the solution can be designed to meet all unique requirements right from the start, rather than extending a commercial product’s core functionality. To quote InfoWorld columnist Polly S. Traylor¹:

“Decades of trial, error, and egghead analysis have yielded a consensus conclusion: Buy when you need to automate commodity business processes; build when you’re dealing with the core processes that differentiate your company.”

Every company likes to believe it is unique. That is what competitive advantage is all about — defining what sets two companies apart from each other when they are competing for the same audience or the same customers. Sometimes what sets a company apart are its wildly nonstandard business or workflow processes. It may seem less painful to accommodate these processes than to change them. That is where customization comes in, regardless of what business need is being addressed by a purchased or constructed solution.

PERCEIVED DISADVANTAGES OF BUILDING: BOTH COST AND CUSTOMIZATION

Even if an organization currently has access to the expertise needed to design and build a solution, a significant disadvantage to building relates to ongoing maintenance and support. Along with a change to the organization’s focus there will be real costs associated with the human expertise and computing resources necessary to maintain a DAM solution.

The purchase of a solution generally includes an ongoing support and upgrade agreement with the vendor. It becomes the vendor’s responsibility to keep up with changing technology and changing business needs within the customer’s industry.

When the decision is made to build a solution, both the team and the organization must realize that they are now responsible for maintaining the solution over its useful life. Even if the original development team quits, that system must be kept up and running and must

be fixed when it breaks. Just for illustration here is a partial list of things that can “break” a DAM system:

- The organization switches to a new page composition software, which introduces another file format that the system must handle.
- A new graphics file format gains wide acceptance.
- The company’s internal networking technology undergoes a major change.
- The company adopts a different desktop environment.
- The company is acquired by (or acquires) a rival and must merge business processes and asset collections together.
- The programming language used to create the system becomes obsolete and the company finds it increasingly difficult to recruit and hire qualified staff.

That list looks rather alarming, but consider that the decision to drive a car comes with its own long list of disadvantages. If we focus only on the risks of accidents, breakdowns, theft and cost of ownership, none of us would ever get behind the wheel. Instead, we balance those risks with the rewards of being able to get where we need to go.

THE INFLUENCE OF ORGANIZATIONAL CULTURE

The decision to build should be made with a full understanding of how the organization “ticks.” This includes its history, business goals and culture. An organization whose culture embraces self-reliance and flexibility and that has a history of adapting to change will stand a greater chance at success with an in-house development project.

That culture must extend to all parties and departments involved with the development project. If the decision to build is made by a “cult of personality,” when that person (or team) leaves it is likely that the project will falter. It is not enough that the main programmer, or IT department, or core user group act as project champions.

Another cultural value that contributes to success is the acceptance that mistakes happen. Projects sometimes fail despite everyone’s best efforts. If the organization’s reaction to missteps is punitive it will stifle any further efforts to

learn from the experience and move on. Fear of punishment is not always the best motivator. Organizations with a more informal, collaborative environment can react more effectively to the setbacks that naturally occur in any large development project. This organizational “personality” tends to be more accepting of an organic rather than linear development approach.

It is vital that all requirements and deliverables are documented and agreed upon by all participants. The milestones, goals and dependencies must be outlined even if timelines are flexible. There is an old adage in the software development world that says the three corners to the development triangle are: fast, cheap and good; you can only have two. Most organizations would do well to aim for cheap and good.

BUILDING WITH OPEN SOURCE TOOLS

The term “open source” is often used interchangeably with “free,” but this is not entirely accurate. The software itself is free, and support may be freely available from other users of the software or from its creator. The GNU project website sums up the definition of “free” in this way:

“Free software is a matter of the users’ freedom to run, copy, distribute, study, change and improve the software.”

Most open source tools are subject to some licensing terms. The most common open source licensing structure is GNU General Public License² or GPL. Another, more permissive license structure is the BSD³ or BSD-style public license. The evaluation process must include becoming familiar with a tool’s licensing requirements.

It is important to remember that with freedom comes responsibility and with responsibility comes expenses. The free software must live somewhere and hardware is not free. In a corporate environment, the programmers who will change and improve the free software tend to ask for things like salaries and benefits. Thus, there are direct and indirect costs associated with the use of open source development tools. The rewards of absorbing

these costs can be enormous for an organization willing to take on the responsibility for the system it creates using open source tools.

THE FREE BITS: SOFTWARE RESOURCES

This section describes some of the tools that can be utilized to assemble an archive of digital photos. This list does not claim to be complete; there are likely other tools that perform the same tasks.

One aspect of the system is the application environment. This includes the operating system, programming and/or scripting languages and the database that will store and index the asset metadata. Here is just one example of a framework that can form the basis of the system:

- Operating system: Linux.⁴
- Web server: Apache.⁵
- Interface development: Django, a Python web framework.⁶
- Programming language: Perl,⁷ PHP.⁸
- Database: MySQL.⁹

TOOLS TO ADDRESS BASIC REQUIREMENTS OF A DIGITAL PHOTO ARCHIVE

This section presents a summary of common technical and functional requirements for a searchable archive of digital photographs. It is not intended to be complete, but rather to present one possible approach. It also references some open source software scripts and software modules that can be incorporated into the archive program.

Digital photographs can be in any of several common formats, including PCD (Photo CD), DCR (a raw camera file format developed by KodakTM), EPS, TIF, PSD. Some files may have been pulled directly from any of several digital cameras, others may have been created by scanning slides and/or positive film.

Available metadata can vary significantly among files, depending on the format and source. For example, TIF files generated by one camera may supply the camera orientation but others may not, and TIFs that were created during a scanning process may lack meaningful camera data. Image resolution data from some formats (PCD, for example) may not be useful

because those files can be opened at different resolutions.

An automated cataloging process would begin with extracting and recording file-specific metadata followed by image processing to create web-ready previews and thumbnails in JPG format.

Metadata can be harvested from the files exif (Exchangeable Image File Format) data. A script called *exiftool*¹⁰ can harvest metadata from a wide variety of file formats. As mentioned previously the amount and meaningfulness of the exif data will vary.

The tools required to create JPG images from the various file formats can include:

- *ImageMagick*¹¹ to convert TIF, PSD and EPS files to JPG. CMYK TIFs require the *lcms* color management engine within *ImageMagick* to ensure that the JPG is properly converted to RGB.
- *hpcdtppm*¹² to convert PCD images to PPM (portable pixel map, a lowest common denominator image format) as an intermediary format, then *ImageMagick* to create the JPG from the PPM. (The PPM files need not be retained.)
- *dcraw*¹³ to convert DCR files to PPM, then *ImageMagick* to create the JPG from the PPM.

For data entry purposes it may be necessary for the user to “zoom” into the image in order to accurately identify objects appearing in the photograph. This can be accomplished by adapting a script called Giant Scalable Image Viewer (*GSIV*¹⁴). *GSIV* is a javascript-based tool for viewing very large images.

As an enhancement to browsing, there is a utility called *similar* from the set of tools known as *Some Tools for Image Collectors*¹⁵ that compares the content of two or more images. It offers several configurations options. This tool gives users another way to browse based on visual comparisons such as color, shape and contrast rather than keywords or structured data.

In addition to development tools and even fully operational DAM software that can be customized under the GNU or BSD license, there are accepted and freely accessible standards ranging from XML schemas (Dublin Core,¹⁶ PRISM¹⁷) to database structures (Fedora Project,¹⁸ DSpace¹⁹) on which to build customized functionality.

Taxonomies may play a role in a DAM solution. The US Library of Congress produced

a freely available tool called the Thesaurus for Graphic Materials.²⁰ The TGM I: Subject Terms list provides valuable raw material for the construction of a hierarchical taxonomy.

THE COSTLY BITS — HARDWARE, EXPERTISE AND TIME

Imagine you respond to an ad offering a free house and what gets delivered are a hammer, nails and a huge pile of lumber. Open source development follows a similar model. The tools are free, but there are direct and indirect costs associated with using them to build something.

Direct costs arise from the tangible objects that must be purchased in order to house the DAM application and/or the digital assets.

Organizations that do not already have sufficient computer hardware and networking infrastructure in place will need to make several hardware purchases:

- *Server(s)*: A basic system can get by with having the database, application and all images stored on the same device but that scenario may not scale up. A large, robust system may require a dedicated database server and possibly also a dedicated image server for low-resolution previews.
- *Storage*: Production-quality digital assets consume vast amounts of virtual space. Any purchase should include room for expansion. It is not unreasonable to start with multiple terabytes and expect to grow from there.
- *Backup mechanisms*: The database, application and assets should all be backed up in some manner so hardware and media add more costs.

Indirect costs arise from the man-hours devoted to analysis, development and maintenance of the software application. These costs start with the analysis phase and will continue throughout the entire useful life of the solution. The development of a DAM system requires several types of expertise. This section describes nine important roles to be played in the development process. These do not necessarily represent individual human beings, but rather a general description of the skills required to pull off a development project.

- *Project manager*: The goals and milestones must be documented and progress towards them must be tracked. This will include bug tracking.

- *Librarian (or Archivist or Information Management Specialist)*: The librarian establishes the “cataloging” process for organizing the digital assets. Expertise with metadata standards and practices is a must. Familiarity with digital archiving standards will also come into play.
- *Analyst*: The analyst will gather and document all requirements, both from a technical and user perspective. The analyst must be familiar with basic DAM concepts, and preferably with several commercial products as well.

The analysis does not end when the system is deployed. There must be a commitment to keep up with technology trends to help ensure that the system remains useful.

- *Programmer*: The programmer will convert ideas into code. A firm grasp of DAM concepts and file formats will be invaluable.
- *Database specialist*: While there are established standards that can be used for guidance, customization and enhancements will be necessary. The database specialist will provide expertise in design and optimization.
- *Interface designer (web designer)*: The designer will help build a usable front-end to the system. Familiarity with web standards and with usability issues will be required. The designer should be comfortable interacting with the development environment, not just with graphical web design tools.
- *Tester*: As the system is developed the tester will help ensure that it works as it should, and that bugs identified and documented.
- *Technical writer*: The system requirements and technical specifications must be scrupulously documented, whether or not the programmer comments his/her code. This includes all the database tables and relationships and the application itself. The writer will also produce and maintain user documentation.
- *Trainer*: Some training will be required when the system initially deploys, but the need will be ongoing as new users come on board, and when enhancements are made to the system.

In a large, formal software development project there may be multiple people in each of these roles (and, in some cases, additional roles such as Architect or Software Designer). Most small, flexible organizations tend to hire people with

multiple skill sets so one person can handle multiple roles. For example, August Home Publishing (the author's employer) is developing a DAM solution with a team of three: a programmer/database specialist, a designer with experience in both communications and web programming and an archivist/librarian with background in technical writing, training and web design. Periodic assistance is provided by the company's network administrator and an outside consultant with Linux and networking expertise.

The time commitment from potential users must also be considered. The DAM system may eventually save time by facilitating access to digital resources, but during the analysis, development and testing phases it can consume time from users who are helping with the process. This can lead to higher-than-expected indirect costs if users' routine productivity is negatively impacted.

CONCLUSIONS

There is no best answer to the question of whether to buy or build an asset management system. Most organizations will find that the only reasonable option will be to purchase a solution that addresses some or all asset management requirements. Others will find that the only viable option will be to build a solution and there are open source development tools to assist in that effort. In some cases a homegrown solution can not only meet some of the organization's asset management needs, but it can also serve as a proof-of-concept to help convince management that a commercial DAM solution would have a measurable ROI.

Disclaimer

This article mentions several software development tools, environments, and applications. Mention does not imply endorsement of any kind by the author or by her employer, August

Home Publishing or any of its partners or affiliates. The tools and applications were freely available at the time of this writing. The author makes no guarantee whatsoever that they remain available at the time of publication.

References and Notes

- 1 Traylor, P. S. (2006) 'To build or to buy IT applications?' *InfoWorld*, 13 February, See: http://www.infoworld.com/article/06/02/13/74880_07FEbuildbuy_1.html.
- 2 GNU license: <http://www.gnu.org/copyleft/gpl.html>.
- 3 BSD license: <http://www.opensource.org/licenses/bsd-license.php>.
- 4 Linux: <http://www.linux.org>.
- 5 Apache: <http://www.apache.org>.
- 6 Django: <http://www.djangoproject.com>.
- 7 Perl: <http://www.perl.org>.
- 8 PHP: <http://www.php.net>.
- 9 MySQL: <http://www.mysql.com>.
- 10 Exiftool: <http://www.sno.phy.queensu.ca/~phil/exiftool>.
- 11 ImageMagick: <http://www.imagemagick.org/script/index.php>.
- 12 hpcdtppm: <http://netpbm.sourceforge.net/doc/hpcdtppm.html>.
- 13 ddraw: <http://www.insflug.org/raw/software/tools/ddraw.php3>.
- 14 gsiv: <http://www.mojavelinux.com/projects/gshiv>.
- 15 stic: <http://stic.sourceforge.net>.
- 16 DublinCore: <http://www.dublincore.org>.
- 17 PRISM: <http://www.primstandard.org>.
- 18 Fedora: <http://www.fedora.info>.
- 19 DSpace: <http://www.dspace.org>.
- 20 Thesaurus for Graphic Materials: <http://www.loc.gov/rr/print/tgm1>.

Further Reading

- Wheatley, M. The myth of open source, (2004) *CIO Magazine*, 1 March, See: <http://www.cio.com/archive/030104/open.html>.
- Oliver, D.(2002) Buy vs. build: Six steps to making the right decision, *TechRepublic*, 22 April; See: http://articles.techrepublic.com.com/5100-10878_11-1038857.html.