Received 4 May 2016
Accepted 4 May 2016

DIGITAL LIBRARIES: THE SYSTEMS ANALYSIS PERSPECTIVE

Organized chaos

Robert Fox
University of Notre Dame, Notre Dame, Indiana, USA

Abstract

Purpose – There are valuable lessons that can be learned from the software industry regarding process improvement. The modern library is now so closely wedded to technology that the way in which services are implemented mirrors that of software development.

Design/methodology/approach – Several methodologies are explored and compared to processes that involve the implementation of digital services in libraries.

Findings – Libraries have reconsidered how they might adopt various business practices to improve service delivery.

Originality/value – This column advocates a study of software development methodologies that have been adopted across the industry with the hope that lessons learned in the corporate world and also in manufacturing could be of value to the digital library.

Keywords Process improvement, Software development, Agile methodology, Iterative development, Software development methodology, Spiral development

Paper type Conceptual paper

It has become almost an obsession in modern culture to perseverate on method. There are many synonyms for this obsession: process improvement, process maturity, continuous improvement, business performance improvement, total productive maintenance, quality management, etc. A contemporaneous bevy of seminars and programs designed to help management teams to enable process improvement has also been spawned that enables this obsession. Examples include the six sigma program and the "Lean" programs that offer certifications or the ability to earn "belts" which are analogous to the levels of advancement common to the martial arts. As an industry, software development processes are also constantly under a certain level of scrutiny to derive the highest levels of efficiency concerning time and effort.

Different types of currencies can be used to quantify various types of work. Currency in this sense is used broadly to indicate a medium of exchange to achieve the optimal outcome. Within the industry, aspects of production such as material usage, hourly wages, overhead and time are examples of the currency used to measure the cost of operations. Libraries have traditionally had a mix of operational work that takes place on a daily basis, most of which involve somewhat complex processes that have for the most part remained static for a long time. Certain operative activities, such as the acquisition on materials, the handling of license agreements, the cataloging and storage of large quantities of bibliographic materials, and the consequent circulation of physical media are all process-based. Although most of these processes will continue into the

Digital Library Perspectives
Vol. 32 No. 3, 2016
pp. 130-136

2059-5816
DOI 10.1108/DLP-05-2016-0012

foreseeable future, it is also clear that the *modus operandi* of libraries has shifted in some cases *en masse* to the digital realm.

Examples of the shift are quite obvious. For instance, the means for consuming academic articles has now almost exclusively become digital, whereas even 10 years ago, this was hardly the case. However, what is not so obvious are the ways in which this shift has transformed much of the daily operational work in libraries. Recent studies have shown recognition of these changes and also a gap between what LIS educational programs teach and what is desired or needed on the job. One interesting 2013 study produced word clouds that visually indicate terms that represent areas where recent LIS graduates either acknowledged an information gap or areas they felt would be critical to library operations within the following five years (Emanuel, 2013). Some of the more prevalent terms in the word clouds were: scripting, programming, XML, databases, cloud, mobile, Web 2.0 and e-books. This indicates that although most librarians will probably do a variety of tasks while on the job, there is a significant trend toward the need to do work that has been relatively isolated to jobs that are characteristically categorized as "information technology". This trend is not isolated just to library specialties such as systems work but broadly for all areas of library work.

As library science becomes wedded to information technology, methodologies in the field need to evolve in an analogous manner. Leigh Estabrook notes presciently in his 2005 article "Crying Wolf: A Response" that:

> Librarianship without a strong linkage to technology (and it's capacity to extend our work) will become a mastodon. Technology without reference to the core library principles of information organization and access is deracinated (Leigh, 2005).

There are signs that the information science field is adopting certain business practices that will be as advantageous for librarianship as they have been for other professions. For example, the discipline of project management is becoming more of a mainstream practice for larger library projects requiring extensive resources, time and effort (Avilés *et al.*, 2015). The need for coordination to meet changing needs is increasing as external forces driving that change accelerate. Because the integration between library science and information technology has become close-knit, this is an opportune time to take a look at strategies that have been adopted by the broader IT community and have been beneficial to that unique workflow.

Prism of methodologies

Since the 1960s, the IT industry has endeavored to perfect the overall process of software development. The reasons for this are not that dissimilar from the reasons that the manufacturing industry has taken a deep look into process improvement over the past 50-60 years. One of the primary goals has been to try and eliminate waste and churn, which can be significant depending on the manufacturing product. The same is true for software implementation cycles. Although manufacturing and software development seem very dissimilar, the need for process improvement is very much the same, and the principles that can be applied are completely analogous.

A paper was published in 1970 by a software engineer from the Lockheed Software Technology Center in Texas, which describes a more typical approach for IT projects. The engineer's name was Winston Royce, and his article was entitled "Managing the Development of Large Software Systems" (Winston, 1970). Royce had been involved

both as a project manager and a researcher in large and complex software projects for the aerospace industry and wanted to share his insights about managing software projects at that scale. The method he described, which was adopted widely by the software industry, was coined the "waterfall" method based on the diagrams that Royce used in his paper wherein the project phases were positioned to look like a staircase proceeding from top to bottom. There are certain phases that are necessary to any software project, whether that involves direct development or even implementation that must be present for the project to be a success. As Royce pointed out, these phases include requirements gathering, analysis, design, coding, testing and, finally, operations. The process outlined by Royce is decidedly linear, which he admitted in the context of the paper. In fact, he outlined one of the major problems with this method that is manifested at the testing phase. By the time a product reaches testing, the development team including designers and programmers has invested a significant amount of effort toward the completion of the project. From their point of view, the portion of work that has reached testing is feature complete and written according to specifications. Testing can reveal two fundamental problems with a product at that point, either the product is written according to specifications but is faulty in the execution and, thus, requires refinement (e.g. bugs) or the product fails to address the original requirements sufficiently and requires a new design. If the problems fall into the latter category, then to a certain degree, the team will need to answer for waste: time and energy that were expended on a product that must be abandoned. Royce states that "In effect the development process has returned to the origin and one can expect up to a 100-per cent overrun in schedule and/or costs". It is easy to imagine what those costs could amount to in the aerospace industry.

Modern librarianship does not suffer from the pressure of cost overruns in the same way that the engineering disciplines do. However, financial costs and time are real commodities that have to be considered regarding new technical ventures that might be undertaken. For example, the implementation of new systems that assist with the acquisitions or cataloging (inventory) process can be costly and requires a tremendous amount of time, for example, time required by budgetary and metadata specialists. These needs have been addressed in the recent past by integrated library systems (ILSs) that handle the majority of automation tasks necessary for library operations. The transition to new systems, or less integrated systems, carries with it substantial migration costs. The phases that Royce outlined still apply. Requirements need to be carefully gathered, the design of the system must be studied, customizations may be applied, bugs will be fixed and library operations are ultimately wed to the new system. The process for engaging in this sort of transition usually simulates a waterfall method and carries with it the same risks.

When the systems that Royce worked with were being developed, there were substantial differences between the methods for automation then and what we can accomplish now. Many similarities exist currently, particularly when complex mechanical, chemical or electrical systems need to be integrated. But, we need to consider the overhead that engineers would have dealt with in the aerospace industry to fully understand Royce's point of view. For example, we know that software systems were used in missiles and aircraft (and still are to an even larger degree). However, the software required to manage those complex pieces of machinery needs to be heavily integrated with physical systems that are of an almost unimaginable level of

complexity. To even determine what can or should be automated, software engineers need to work closely with scientists and other engineers who need to consider all physical factors involved in developing those machines. The inputs and outputs for those systems need to be considered very carefully and should be designed with maximum precision. So, the initial requirements, analysis and design phases have an elongated scope because of the intricacy of the subject. Using the waterfall method described in Royce's paper may, therefore, be appropriate for certain development environments where a high degree of integration or complexity is involved.

Complexity of the subject matter is one factor that can be used to determine the optimal methodology for IT projects, but it is one among many. Several other issues have necessitated other approaches that vary in flexibility and commitment. These have evolved over time to address new situations that have arisen in the software industry to achieve a more complete level of compatibility between the needs of software developers, consumers and product owners. As noted earlier, the problems that software is trying to address may bring with them a very high level of complexity. However, as software has become more mature, the inherent complexity of software itself has grown in a commensurate manner. That inherent complexity drives the need to think of larger software projects according to their components as opposed to the application as a whole. This is also true for the more involved implementation projects such as the ILS migration project mentioned earlier.

When larger projects are broken up into smaller components, it affords the team the ability to treat those smaller components separately and proceed with micro-development cycles that are concerned with just those components. Several methodologies have grown out of this notion and are labeled in various ways, including prototyping, incremental development and iterative development. The core idea with these methodologies is having the micro-cycles running concurrently where design, coding and testing are done in rapid succession. The feedback cycle allows the team to perfect one aspect of the overall application without interfering in the work of the other aspects. This method also allows the developers or operational staff to interact with users and product owners on a regular basis. The positive effect of this is that the team can presumably avoid the situation described by Royce as involving cost overruns because the corrective measures can be incorporated during development instead of at the conclusion.

A close cousin of the iterative method is the spiral development method. This also involves tight cycles of planning, development, testing and requirements gathering, but the differentiating factor is risk assessment as each iteration takes place. Each cycle brings with it several key indicators: the cumulative cost, prototypes for testing and user engagement and further planning and review. At any point, the risk level may indicate a varying course of action depending on the situation with the ultimate goal being final implementation and release.

The latest evolution of iterative development came at the dawn of the twenty-first century with the advent of agile software development. The primary driving factor of the agile method is change. It embraces the need for iterative cycles and close communication with product stakeholders and anticipates change by building in flexibility. Prototypes are a key aspect of the agile method, but components are broken down even further into features. The cycles of the phases outlined earlier are also shortened, usually to a small window of time. Each team determines what that window of time will be, and, by team, the agile method is referring to the product owners (or

sponsors), the developers, QA analysts, usability experts and the users themselves. All stakeholders are included in the review phase, and the extremely short cycles are designed to provide maximum mitigation against risk. Course corrections can be made very quickly.

As previously stated, the agile method breaks applications, components and larger aspects of the application down into feature sets. Flexibility is built into the cycles through the focus on a minimum viable product for any given feature. Once the minimum functionality is achieved, the team can proceed with feature enhancement. This evolutionary way of writing software while building in feedback is almost completely opposed to the waterfall method that was first described. In the waterfall method, the end product is meticulously defined, and the outcomes and expectations are set in stone as it were. The deliverables cannot be modified over time using waterfall because it would undermine the previous phases in the process. This is effective when nothing changes, including the environment, further research, the user base and other factors. The assumption of the agile method is that this is never the case. If the opposite is assumed, then waste is practically guaranteed.

The Agile Manifesto[1] was written in 2001 by 17 software developers of high caliber who had extensive experience working in the corporate sector as software engineers and who also were proponents of lightweight methods for producing better software products. The Manifesto is really a set of specific principles such as "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation" and "Simplicity – the art of maximizing the amount of work not done – is essential". Keep in mind that these platitudes were born out of years of experience and lessons learned. The true heart of agile thinking, though, is the focus on the consumer of software. This is stated in a nutshell via the manifesto itself:

> Individuals and interactions over processes and tools.
>
> Working software over comprehensive documentation.
>
> Customer collaboration over contract negotiation.
>
> Responding to change over following a plan.
>
> That is, while there is value in the items on the right, we value the items on the left more.

The agile mindset has been adopted by organizations with varying levels of success. The original authors would probably agree wholeheartedly that the principles outlined in the manifesto are more important than dogmatic adherence to a particular way of doing business. Any method needs to be adapted to an environment to make it work. In fact, this is one of the core principles of the agile mindset.

Focused and flexible

Although the principles outlined above have grown out of the general IT workflow, it is clear that they can also have a huge impact on the digital library. Any aspect of library operations can benefit from process improvement, though the environment for large and small libraries has changed. The gap between the customer or patron and the work of service to those customers has diminished rapidly because of technology. Recalling what was said earlier about the various necessary phases of software development, we

can see clear analogs in information science. There is both a symbolic and actual parallel between service delivery; content delivery and access; and product development. From a symbolic perspective, the services provided by a modern academic library are the core deliverables we provide to patrons. From the concrete perspective, technology and software applications are the typical means by which those services are made accessible.

A recurring aspect of the methodologies outlined previously is that of iteration. This is not a reference simply to practicing repetition until, almost by accident, the outcome is positive. The recursive nature of rapid prototyping, spiral development and agile methodologies is very intentional regarding the focus on continuous communication and feedback. Assumptions have been made that vendors, or software developers, or even user experience analysts always have the right answer when it comes to breaking down barriers between patrons and information provided by the library. This is a wrong-headed assumption on several levels. First, every environment and institution is unique. The values and the needs of those doing research are not monolithic and vary according to institutional focus. Second, external parties are not in regular contact with the true customer of library services, namely, the patrons. A vendor will view the librarian as the customer. A software engineer will do the same under many circumstances and in isolation. The proven methods of implementing digital services and software products do not adhere to rigid processes or misplaced trust. Those processes that truly work involve collaboration, and they must be able to adapt to changing circumstances.

As with the agile manifesto, librarianship in the digital realm has a set of core principles that cannot be abandoned. However, the means by which those principles are embodied in products and services need to be flexible to remain relevant and effective. A key aspect of being flexible is the adoption of techniques for software implementation and development that encourage a tight feedback loop that can be acted upon. One of the complaints of the manifesto authors was that dogged adherence to process got in the way of communication. The same can be true in the library environment when the patron is not consulted during important instances of decision-making. The feedback loop has a tendency to be limited to the library and the software vendor, or the library, and the software engineer or a combination thereof. This simulates the artificial barriers constructed in the corporate setting where the project plan, initial requirements (or worse, assumed requirements) and a timeline is doggedly adhered to without reference to changing needs. It is always important to keep focused on well-intentioned goals, but it is equally important to remain flexible and responsive *vis-à-vis* the ultimate consumers of digital library services.

Note

1. See http://agilemanifesto.org/

References

Avilés, R.A., Marco-Cuenca, G., Serrano, S.C. and Simón, L.F.R. (2015), "Project management in library and information science: an application in university library context", *Handbook of Research on Effective Project Management Through the Integration of Knowledge and Innovation*, pp. 405-1006.

Emanuel, J. (2013), "Digital native librarians, technology skills, and their relationship with technology", Digital Services & Reference Librarian, University of Illinois, Urbana, *Information Technology & Libraries*, Vol. 32 No. 3, pp. 20-33.

Leigh, S.E. (2005), "Crying Wolf: a response", *Journal of Education for Library & Information Science*, Vol. 46 No. 4, pp. 299-303.

Winston, W.R. (1970). "Managing the development of large software systems", Technical Papers of Western Electronic Show and Convention (WesCon), 25-28 August 1970, Los Angeles, CA.

Corresponding author
Robert Fox can be contacted at: rfox2@nd.edu