Communications of the Association for Information Systems

Volume 30 Paper

3-2012

Effective Application Maintenance

James D. McKeen
School of Business, Queen's University, jmckeen@business.queensu.ca

Heather A. Smith
School of Business, Queen's University

Recommended Citation
McKeen, J., & Smith, H. A. (2012). Effective Application Maintenance. Communications of the Association for Information Systems, 30, pp-pp. https://doi.org/10.17705/1CAIS.03005

Communications of the Association for Information Systems

CAIS

Effective Application Maintenance

James D. McKeen

School of Business, Queen's University

jmckeen@business.queensu.ca

Heather A. Smith

School of Business, Queen's University

Abstract:

Application maintenance is a critical, yet unheralded, activity in most IT organizations. Left unmanaged, maintenance costs can easily spiral out of control to consume the lion's share of the IT budget. As the saying goes, "If you don't manage maintenance, it will manage you." To explore the organizational issues of application maintenance, the authors convened a focus group of senior IT managers from a variety of different companies representing several industries. This article is based on these discussions. It first examines the root causes of application maintenance and the importance of this issue. It then reviews the sourcing–maintenance debate and explains the "maintenance trap" that has plagued IT organizations for a long time. Application support is then investigated and linked to the four types of maintenance (i.e., corrective, adaptive, preventive, and perfective). The article concludes with a number of proven strategies suggested by the focus group to enable IT organizations to manage application maintenance … rather than the other way around.

Keywords: application maintenance, application cost, application design, application lifecycle

Volume 30, Article 5, pp. 73-82, March 2012

Communications of the Association for Information Systems

I. EFFECTIVE APPLICATION MAINTENANCE

For many years, organizations have been following the path of automation. What started with repetitious tasks typically in accounting has now spread to all aspects of modern-day organizations. Organizations have automated processes such as supply chain management (SCM), customer relationship management (CRM), human resource management (HRM), and financial management. Beyond these basic capabilities, there are myriad automated functions unique to industries and individual firms. Early, perhaps rudimentary, automation has given way to highly sophisticated types of automation that delivers efficiencies and capabilities to organizations and helps to differentiate them from competitors in the marketplace.

Automation of an organizational process harnesses information technology, usually by acquiring or developing a new application. As a result, the extent of automation is represented by the size and complexity of the portfolio of applications that "run the business." The longer an organization has been traveling on the path of automation, the greater its application portfolio. Many applications are now old (commonly referred to as *legacy),* have undergone countless changes over their existence, run on technologies that are no longer supported, and, as a result, are quite fragile. But, regardless of their circumstances, all applications need to be maintained in order to remain productive. Therefore, as automation proceeds and the application portfolio grows, so too does the maintenance budget, and organizations can easily wake up to discover that virtually all of their IT budget and resources are consumed by activities that simply "keep the lights on." As the saying goes, "If you don't manage maintenance, it will manage you."

To explore the organizational issues of application maintenance, the authors convened a focus group of senior IT managers from a variety of different companies representing several industries, including manufacturing, insurance, consulting services, banking and finance, retail, and telecommunications. In preparation for the meeting, focus group members were asked to respond to a series of questions, including: What is your annual expenditure on maintenance (e.g., number of staff, percentage of budget)? Are these costs trending up/down? What are the key drivers behind the size of the maintenance and support effort in your organization? What strategies have you implemented to deal with the maintenance issue? How successful have these strategies been?

The group was sequestered for an entire day, and each member of the group shared his or her organization's response to the above questions. The discussion was moderated by one of the authors, while the other author took notes to capture the key discussion points. Focus group members also shared organizational policies and procedures in order to explain how their organizations were attempting to manage the demand for IT services. This article is based on the content of the focus-group discussion. We first examine the root causes of application maintenance and the importance of this issue. We then review the relationship between support and maintenance and present four types of maintenance. We conclude with a number of proven strategies suggested by the focus group to manage maintenance effectively.

II. UNDERSTANDING MAINTENANCE AND ITS CAUSES

What exactly is application maintenance, and why is it necessary? Theoretically, there is no need for applications to be maintained. An application does not deteriorate over time. There is no friction ... no wear and tear. There is no force of nature tugging an application toward entropy. In contrast to the physical world of maintenance, application maintenance arises from the desire to change what an application does, how it does it, or the environment within which it operates. Porting an application to a new server, interfacing with a different operating system, upgrading to a newer release, altering a tax table, or complying with new regulations—all necessitate application "maintenance." As a result, maintenance is focused on upgrading an application to ensure it remains productive and/or cost effective. The definition of application maintenance preferred by the focus group is "any modification of an application to correct faults; to improve performance; or to adapt the application to a changed environment or changed requirements." Thus, adding new functionality to an existing application (i.e., enhancement) is not, strictly speaking, considered maintenance.

Keeping an application productive (i.e., maintaining it) requires ongoing attention over the lifespan of the application. Focus group members pointed out that the lifetime costs of maintaining an application typically exceed its original acquisition costs many times over. Various estimates exist about the costs of maintenance and support. One source claims, "Large enterprises spend an average of 73% of their IT budget on application maintenance of existing systems" [Zensar, 2011]. McKeen and Smith [2009] studied an organization where maintenance represented 46

percent of their overall IT budget. Without questioning these numbers, the point can be safely made that application maintenance and support consumes a very significant percentage of IT expenditures and is, therefore, an activity that, if managed aggressively, has the potential to save considerable amounts of money and resources. This fact alone makes maintenance a perennial management issue for organizations.

When it comes to maintenance, not all applications are the same. Some require little attention, while others are maintenance "hogs." According to one focus group member, many of the high maintenance applications in her organization are poorly designed. Her experience suggested that simply redesigning an application could significantly reduce the amount of maintenance required. For example, a data- or table-driven application is superior to a code-driven application that requires code changes for even minimal modifications. The focus group identified a number of characteristics of applications that collectively dictate the level of maintenance required. Most of these factors relate to the history of the application and the quality of its design.

1. Siloed Development
   For many years, applications were developed within business silos. Over time, these legacy applications have generated significant value based on a multitude of enhancements and new features [Vibbert, 2011], but maintaining these often-redundant applications demands extensive resources. One focus group member cited the example in her organization of the proliferation of "client server environments with little architectural structure, few defined processes, silo builds and multiple platforms."
2. Layers of Technology
   Over time, applications become "layered" in technology. A basic practice is to use new technology to preserve old technology—a practice referred to as putting "old wine in new bottles" by the focus group. Newer and more efficient technologies can be made to emulate older technologies just as middleware can integrate different generations of technology by decoupling them. Well-layered and modularized applications can actually reduce maintenance, but poorly layered applications can substantially increase the costs of maintenance.
3. Service Enablement
   It is becoming common practice to service-enable legacy applications in order to modernize and integrate them with newer applications. Functional code, as well as data, are extracted from the original legacy application and wrapped within a defined interface so that its functionality can be used by other applications. This level of modularization should, at least theoretically, reduce the complexity of an application.
4. Age of Code
   Although common wisdom suggests that the older the application, the higher the required maintenance; the focus group did not accept this adage. They cited many counter examples of older legacy environments (e.g., OS390 and AS400) that are mature, have well-developed processes, are well-architected, run forever, and experience lower maintenance costs with age. Their argument was that older applications, if maintained properly, could be more stable and easier to support than newer applications that tend to have more issues. Certainly, the age of an application does not cause maintenance in and of itself; however, as applications undergo significant modifications over their productive lives, they can become increasingly complex and, therefore, more difficult to maintain.
5. Lack of Documentation
   The lack of adequate documentation renders applications difficult to modify. It encourages reliance on individuals with extensive knowledge of particular applications. Without these individuals, maintenance activities consume substantial resources, sometimes just to determine how the application works.
6. Frequency of Change
   Applications that have not undergone substantial revisions over time remain in a pristine state. These relatively unchanged applications are easier to maintain because they remain largely intact. For example, a currency conversion application at one organization had not been touched in decades.
7. Work-arounds
   When changes are made to applications, sometimes shortcuts are taken—commonly referred to as "work-arounds." Although such practices may save time and money in the short run, they can exact severe penalties in the long run as they cascade (i.e., work-arounds on top of work-arounds). The resulting application becomes a quagmire and even the simplest of changes soon becomes exorbitantly complicated.
8. Complexity
   Some applications are more complex than others simply by the nature of their functionality. These will always cost more to maintain. Complexity is also driven by any combination of the above factors. For example, applications that have undergone multiple changes, generations of technology, countless work-arounds, and inadequate documentation will undoubtedly be highly complex and, therefore, expensive to maintain regardless

of their functionality. One manager suggested that her firm's "recent adoption of portals, web instances, and a boom of different components have further increased complexity."

In addition to these application characteristics, the focus group identified three organizational factors that also affect maintenance. The first factor is the organization's problem resolution culture. One member of the focus group explained it this way: "When the application maintenance and production support culture is one of "symptom fixing," it can generate more maintenance issues in the long-run instead of reducing them. Having a culture that does root cause analysis for each problem found eliminates related future problems before they happen, thereby reducing maintenance in the long-run."

The second factor is software platforms. According to the focus group, some software platforms have been proven more reliable than others (e.g., UNIX versus Windows versus Oracle). As a result, the choice of platform can have a material impact on the downstream maintenance on hosted applications. The unreliability of a platform renders applications unstable and complicates troubleshooting of application errors because the platform must be ruled out as the potential culprit first.

The third organizational factor is technology diversity. Maintenance is directly related both to the number of applications involved and to the diversity of applications. The number of different components in an organization's technical environment (e.g., platforms, operating systems, databases, test environments, supported devices) dictates the number of full-time equivalents (FTEs) to support it, as well as the type of skills to retain, the vendor relationships to be managed, and, therefore, the overall costs of maintenance. Organizations that can streamline their technology environments can certainly reap significant savings on the maintenance function.

III. THE SOURCING-MAINTENANCE DEBATE

During this discussion of maintenance, the focus group also explored the issue of application sourcing. The question posed was: Does outsourcing or buying vendor applications result in less expensive and more reliable applications than developing such applications in-house? Three popular sourcing alternatives were considered: customer-developed software/applications (CDS),[1] commercial off-the-shelf (COTS), and software as a service (SaaS). Each alternative was examined from a total cost of ownership (TCO) perspective that included the maintenance costs:

- CDS applications (whether in-house or outsourced) incur initial development costs, run costs, base essential support costs, and ongoing maintenance.
- COTS applications incur initial licensing, ongoing licensing and maintenance, run costs, and base essential support.
- SaaS applications incur an annual vendor cost (typically user-based), base essential support, and ongoing maintenance of interfaces/data exchanges.

The conclusion reached by the focus group was that maintenance is universal and vendors have the same software quality issues as any other software development group. When making the decision to outsource, use a vendor, or build in-house, the quality and reliability of each option must be verified and weighed thoroughly. Every software development group has its strengths and weaknesses, regardless of whom they are employed by. Members cited examples of needing to in-source vendor source code, as well as examples of packaged systems that turned into maintenance nightmares, especially when the vendor went out of business or was taken over. In-house developed applications were just as apt to become maintenance problems. The key is to evaluate and assure the software development quality regardless of source. One company formed a quality assessment team to visit potential vendors in order to authenticate their quality and reliability in producing applications.

In summary, all applications require maintenance regardless of how (or by whom) they are developed. The only real difference, which is actually not germane to the discussion of the nature of application maintenance, is whether or not maintenance costs differ, depending on the sourcing model. At least theoretically, a provider of commercial applications should be able to amortize the maintenance costs over a broader base of customers and thus reduce the maintenance costs for all customers. This assertion, however, was not substantiated by the focus group, based on their collective experience.

[1] A customer-developed application can be developed in-house or it can be outsourced. The key feature of this type of development is that it is bespoke; that is, it is customer-tailored to fit specific business requirements.

IV. THE MAINTENANCE TRAP

The maintenance trap is best explained by the following example that was described by a manager within the focus group:

> *A business manager asks for a self-serve capability to be added to an existing HR application. The IT manager determines that this change would cost $80,000. However, the IT manager is also aware that the HR application is poorly structured, is not fully documented and contains a number of known "workarounds." She determines that all these flaws could be remedied for about $20,000 if done while adding the self-serve capability. Furthermore, correcting these flaws would result in a technically healthier HR application that would save the organization money in the future by prolonging its productive life. Acting in the best interests of the organization, she tells her business manager that the new self-serve capability would cost $100,000, reasoning that her business partner need not know her plan to refurbish the HR application. Using the $100,000 estimate, the business manager creates a business case, wins approval and implements the self-serve capability.*

In this scenario, everyone appears to win; employees benefit from a new self-serve capability and the organization benefits from a technically rejuvenated application. Unfortunately, the need for (and the costs of) maintenance has been hidden from management. So IT's *dirty little secret*—the term coined by a member of the focus group—confirms to management that it is unnecessary to spend on application maintenance. The upshot is that the business will not fund application maintenance [Swanton and Kyte, 2010] which forces the IT organization to secretly cloak it under enhancement. An analogy is the car mechanic who replaces some worn hoses unbidden while he has the hood up on your car. Replacing worn hoses and refurbishing an HR application is NOT the problem; the problem is taking these actions without full disclosure to the decision maker—the car owner or the business manager.

No business manager would question the need to maintain a truck fleet or a physical plant. Yet they have "learned" (by means of the "maintenance trap" described above) that IT applications seemingly exist without the need for maintenance. Furthermore, they do not see any contradiction in this differential treatment of assets—application assets versus physical assets. The view of the focus group was that IT organizations bear responsibility for creating and condoning this situation, and, therefore, they are responsible for finding a way to escape this trap. Until then, organizations will be unable to deal effectively with application maintenance.

V. MAINTENANCE VERSUS SUPPORT

According to Kyte [2010], there are four different types of maintenance, as shown in Figure 1 and explained below.

1. Corrective maintenance

   Despite thorough testing, every application experiences errors after implementation. When errors are detected, they must be fixed, which is why this type of maintenance is commonly referred to as "break/fix." Errors run the gamut from "nuisance" to "critical" and must be triaged by the support team (described below).

2. Adaptive maintenance

   Applications run within a highly complex environment. When upgrades are made to elements within this environment, this often necessitates changes to the application. For example, an application may be migrated to a different database (or server) to enhance its operation characteristics.

3. Preventive maintenance

   This is forensic activity to detect the presence of errors before users encounter them. When applications are first released, there is a high incidence of errors. Rather than wait for these to be detected through actual usage (which may cause disruption to the business), preventive maintenance deploys root cause analysis to isolate these errors and apply corrective measures.

4. Perfective maintenance

   Applications are designed to meet both functional requirements (e.g., initiate a loan) and nonfunctional requirements (e.g., efficiency). The goal of perfective maintenance is to improve the performance of an application, so it normally targets nonfunctional requirements that often change over the lifetime of the application. For example, an application may require perfective maintenance to reduce its average response time.

The support function, also depicted in Figure 1, plays a central role regarding the maintenance of applications. There are typically three levels of support. The first level is the helpdesk where users first go to for any problem, outage, or difficulty with an application. The first level responder would be assisted by a knowledge-management system that

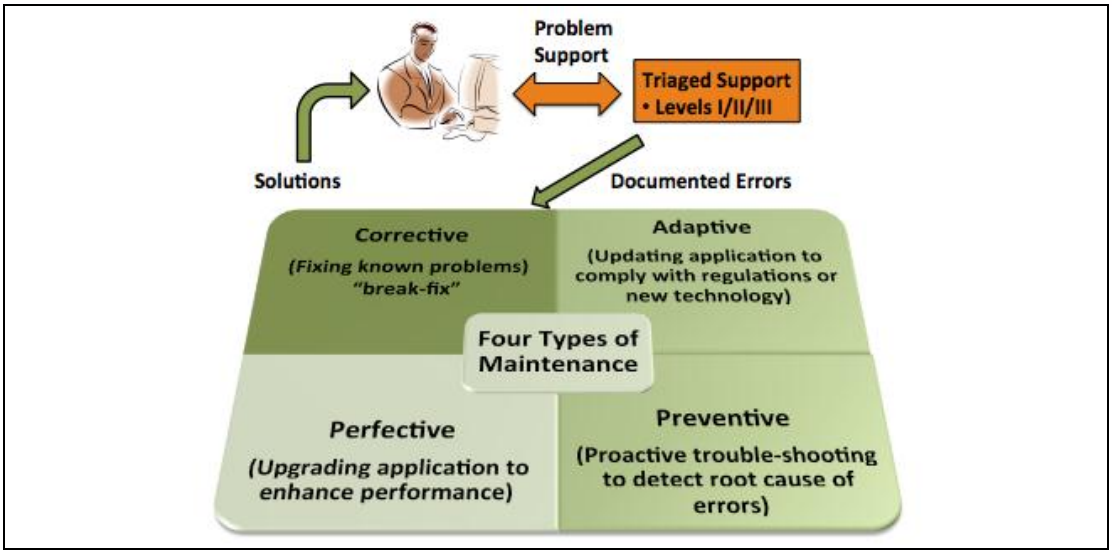

Figure 1. Relationship Between Maintenance Support

recommends some basic actions to try to resolve the issue. If that is not successful, the issue is referred to the second level of support where people have specialized knowledge about the target application and should be able to resolve most issues. If that is not successful, the issue is referred to the third level of support, which is often the application developers, architects, or vendors (for externally supplied applications).

Typically, organizations offer different levels of support, depending on the criticality of an application. Higher levels of support provide more resources per application, maintain more knowledge about the application, provide subject-matter experts, and are able to respond faster and more capably at an increased cost to the business. For less-critical applications, fewer resources would be available, response times would be slower, and resolution of issues may take longer. This level of support would, of course, cost less.

The support function not only assists corrective maintenance by identifying errors in applications, but also plays an important role in perfective maintenance. Because the helpdesk provides first-line assistance to the business, it is in an advantageous position to identify troublesome aspects of applications. For example, business users may be frequently confused by the same input screen, resulting in a high number of calls to the helpdesk. While not erroneous, this part of the application would certainly benefit from redesign as part of perfective maintenance. The result would be fewer calls to the helpdesk, less frustration experienced by business users, and increased productivity across the organization.

Whether applications are developed internally or sourced externally, the focus group argued that the maintenance and support functions are basically the same. The only difference is who provides the service. For instance, for an application service provider (ASP) application, the vendor owns, runs and maintains the application and is responsible for providing helpdesk support and all four types of maintenance. Internal IT staff may (at their discretion) manage problem incidents in order to oversee issue resolution, but even this can be outsourced.

Finally, preventive maintenance must be distinguished from testing. Testing is "preventive" in that its goal is to detect and correct errors before they impact the business, but the key difference is that testing happens before an application is put into production and preventive maintenance happens post-release. The other difference is in the techniques that are used. Testing involves simulating a production environment complete with realistic test data. Techniques used in preventive maintenance tend to be more forensic and include practices such as tracking individual programmers known for their higher error rates, root cause analysis, and pattern-based error detection.

VI. SUCCESSFUL STRATEGIES FOR MANAGING APPLICATION MAINTENANCE

The focus group suggested a number of strategies for improving how maintenance is managed, based on their collective experiences. Some of these strategies constitute prevention (i.e., actions that avoid/reduce the need for maintenance); others entail correction (e.g., effective ways of performing maintenance); and others are generic (e.g., effective management). All strategies are related.

1. Prioritize Applications for Maintenance

   Applications are not equally critical to the ongoing functioning of the organization. Those that are mission critical require 24/7 support by a team of highly skilled staff; problems with less critical applications can be satisfied with a next day or end-of-week response. Applications should, therefore, be prioritized in terms of their criticality to the business in order to determine the appropriate level of maintenance and support. Experience among our focus group members suggests that two to four prioritization categories are effective (e.g., critical versus noncritical; gold versus silver versus bronze). Each application category establishes the level of maintenance and support that is available and the associated cost to the business. In the absence of such a prioritization scheme, requests for maintenance and support are handled on a "first in ... first out" (FIFO) basis, which results in situations where critical applications wait for resources, while noncritical applications receive priority treatment. This lack of prioritization results in elevated maintenance and support costs and overall poor service.

2. Design for Maintainability

   Most application errors are easily traced back to faults at the test stage (i.e., how did this error escape detection during testing?) and, more importantly, at the design stage (i.e., why was the application not designed more effectively at the outset?). Some applications are notoriously error-prone. As previously identified, sometimes this is due to an application's history (e.g., existence of multiple work-arounds), and sometimes it is due to poor design. The focus group stated that the lifetime difference in maintenance costs between a well-designed application and a poorly designed application can be orders of magnitude. Therefore, it behooves IT organizations to enforce rigorous design standards.

   There is no doubt that the first line of defense against corrective maintenance is design; a well-designed application produces fewer errors. In addition, a well-designed application aids adaptive and perfective maintenance. Effective design yields error-resistant and change-ready applications. One manager claimed that the adoption of the Capabilities Maturity Model Integration (CMMI) practices by her application development group has improved their IT-related processes and raised the overall quality and reliability of their IT applications substantially for all applications (i.e., in-house and outsourced).

   Most development groups are aware of effective design principles. So the problem is not ignorance; rather, it is adherence to (and enforcement of) design principles. Design guidelines provide no benefit if they are not followed. The culprit, according to the focus group, is the attempt to satisfy harsh deadlines and/or excessive cost pressures, both of which entice developers to "cut corners." The solution is to instill a culture within the IT organization from the very top that does not allow exceptions to established design standards when developing applications. Sometimes business pressure mandates that corners be cut in order to meet market deadlines. In these (hopefully infrequent) occurrences, the focus group suggested that the business must assume responsibility for making such decisions with a full understanding of the ramifications. At one organization, circumventing design standards designates the application as "at risk," which means that the business can expect to face higher maintenance charges in the future until/unless the application's design shortcoming are rectified. Although the promise of reaping immediate gains is tempting, the result is always long-term expense.

3. Test for Maintainability

   Testing procedures are well-established in most IT organizations. These include multiple types of testing (e.g., unit, integration, user acceptance testing), partitioning of environments (e.g., development, test, and production environments), as well as specialized testing (e.g., audit, compliance, and risk). The focus group advocated that applications should also be tested for maintainability. This checkpoint would be conducted by a team of IT personnel with technical skills sufficient to verify that an application has followed all the design rules and guidelines established by the IT organization. This would be one of the last checkpoints before the application is declared eligible for production. One company tasks the maintenance team with performing this test. The logic is that they have to maintain the application throughout its productive life so it is in their best interests for applications to follow a well-prescribed design template. The need for this "extra" sign-off was originally questioned by the business, but the company was able to justify this action based on an analysis of existing application costs provided by their application portfolio management (APM) initiative [McKeen and Smith, 2010].

4. Modernize Your Legacy Applications

   Legacy applications are both an asset (because they run the business) and a curse (because they become stumbling blocks for organizations attempting to keep up with the ever-increasing demands of competition). In addition, legacy applications can cause major disruptions within a company's application ecosystem because they use outdated technologies, multiple user interfaces, or multiple security frameworks. For these reasons, it is crucial that such applications be modernized if an enterprise wants to stay productive and competitive.

Modernization of legacy applications differs from corrective maintenance (since the task is not to correct errors), adaptive maintenance (since modernization focuses on the application itself and not its environment), preventive maintenance (since it is not focused on error elimination), and perfective maintenance (since the task is not simply performance-enhancing). Instead, legacy application modernization focuses on re-architecting applications so that future maintenance is reduced and they become change-ready.

Legacy application modernization also involves application rationalization to address the duplication problem. Due to historical silo development, it is not uncommon for users to be forced to employ several legacy applications to perform the tasks within a given business process. According to one manager, such situations present ideal opportunities for overhauling user interfaces in order to enhance access and productivity. At her organization, a group of legacy applications with multiple access points was transformed to enable a single view into the entire system via social media software, business process management systems, and business intelligence tools.

A final approach suggested by the focus group for application modernization is the combined usage of Web services, middleware, and reuse. These tools allow multiple legacy systems to be integrated behind the scenes through a combination of decoupling and delayering legacy applications into key components that can be reused. With an effective abstraction process, legacy applications don't have to be completely redesigned.

1. Assign maintenance based on criticality and career aspirations.

   There is an ongoing debate whether maintenance should be handled by permanent maintenance teams or by development teams. Permanent maintenance teams benefit from specialization by honing general skills such as troubleshooting and diagnostics. Development teams have specialized knowledge of the inner workings of the applications they developed. Both skill sets are valuable. The focus group argued that there is no best answer to this question. Instead, they argued that the decision should be driven by the criticality of the application and the career aspirations of the IT professionals. The reasoning is that mission critical applications need skilled personnel available 24/7 and IT professionals typically gravitate toward a distinct preference for either development or maintenance. In structuring maintenance activities, therefore, the guiding principle is to align the career aspirations of IT personnel with applications depending on their criticality. This strategy avoids the negative perceptions of maintenance by creating a dual stream for career development within IT.

2. Make maintenance transparent.

   Business managers own applications. For this reason, maintenance must be transparent and understood by these business managers so that well-informed decisions regarding the disposition of these assets are forthcoming. Although it may be pragmatic to advocate the practice of funding maintenance from "skunk works" [Swanton and Kyte, 2010], the focus group strongly recommended against this practice. Their primary objection was that this practice hides maintenance (and hence its need) from the business owner (described earlier as the "maintenance trap"). Although it gets the job done (like the mechanic replacing hoses before they break), it leaves the owner out of the decision process. This perpetuates a vicious circle where business managers resist funding maintenance which forces IT managers to use their slush funds.[2]

   To break this circle, maintenance and the need for regular maintenance must be demonstrated to business management for two reasons: first, as owners, they need to be involved in all investment decisions that impact their applications; and second, maintenance costs are material. According to the focus group, maintenance costs over the lifetime of an application typically dwarf the initial development costs.

   The focus group suggested a number of strategies to make application maintenance transparent. They recommended that IT organizations adopt full lifecycle costing methods for key applications. To do this, future costs are estimated for each of the four maintenance types (i.e., corrective, adaptive, perfective, and preventive). Techniques exist for accomplishing this (e.g., see Kyte, 2011). With these estimates, application decisions can be based on comprehensive information, as opposed to a succession of piece-meal decisions over the lifetime of the application. The other key strategy suggested by the focus group was to tie maintenance costing into an existing application portfolio management (APM) initiative. Adding maintenance information to the information that is already being captured for applications can be accomplished with modest incremental cost and huge downstream benefit.

[2] According to the focus group, slush funding (i.e., skimming off other development projects with committed resources) presents additional complications. This practice, when done judiciously, does not jeopardize the delivery of these development projects, but it does obfuscate the true costs of these projects. In the end, maintenance is hidden from the business, so it appears unnecessary. It also appears "free," while other development projects appear "expensive." This distortion serves no one and should not be condoned.

3. Formalize governance procedures for maintenance.

   The backlog of maintenance requests is infinite and represents years of work. Indeed, new applications are often launched with extensive lists of required changes. As a result, maintenance needs to be managed aggressively so that it doesn't grow to consume all available IT resources. To do this, well-designed governance procedures must be developed and implemented to govern maintenance activities. The focus group suggested the following key procedures.

   - Establish standards to guide maintenance.

     Without maintenance standards, as errors are fixed, new ones can be introduced [Kyte, 2011]. This affects the stability of the application code and renders future maintenance difficult and expensive. All maintenance activity must follow the established guidelines that cover documentation, coding, sign-offs, program structure, and format. One manager stated, "We want standard operating procedures so all 30 BAs do it the same way."

   - Establish testing procedures.

     In order to verify that errors have been corrected and/or new changes function correctly, standard test environments and procedures must be established. These test environments and assets must be maintained as the application is enhanced and extended over its productive lifetime. This practice enhances the quality of the testing procedures and reduces future testing costs by providing an updated and comprehensive testing environment.

   - Formalize request for change (RFC) procedures.

     Maintenance standards and guidelines should include formalizing procedures for handling requests for changes. At one organization, the IT organization operated as an "order-taker" to the business, and, as a result, attempted to respond to each request for change on a FIFO basis. New changes were "put through each night." This organization has now adopted a formalized procedure for handling maintenance requests which must be documented, approved, and prioritized by the business before the IT organization schedules the work. Although this requires extra steps, it ensures that the most important maintenance activity gets done first.

   - Control release management.

     The focus group suggested that there are advantages to bundling maintenance requests and managing new releases. There are many reasons for this practice. On the technology side, enacting multiple changes to the same application simultaneously saves resources like coding, testing, documentation, and versioning. Bundling also facilitates business cycles by scheduling new application releases to avoid peak business periods. One manager claimed that his organization's ability to plan and schedule maintenance has also made it easier to handle the unexpected "emergency changes" and "fire fighting" activities.

VII. CONCLUSION

The IT application portfolio must continually change in order to meet the demands imposed by the business in order to stay competitive. This requirement puts enormous strain on the IT organization, not the least of which is the escalating costs associated with maintaining the IT application portfolio. Without careful attention, these costs can quickly grow to consume all available IT resources, thus reducing, if not eliminating, discretionary IT investments. The solution is to actively manage maintenance to ensure that IT applications remain productive in their role of supporting the organization's activities. This article looked at the nature of application maintenance and its root causes, reviewed the different types of maintenance, and examined the critical relationship between support and maintenance. Based on experiences of the focus group, a number of strategies were presented to successfully manage IT application maintenance. Maintenance has become a critical management activity for organizations; therefore, managing it effectively should be a top priority for IT leaders.

REFERENCES

Editor's Note: The following reference list contains hyperlinks to World Wide Web pages. Readers who have the ability to access the Web directly from their word processor or are reading the article on the Web, can gain direct access to these linked references. Readers are warned, however, that:

1. These links existed as of the date of publication but are not guaranteed to be working thereafter.
2. The contents of Web pages may change over time. Where version information is provided in the References, different versions may not contain the information or the conclusions referenced.
3. The author(s) of the Web pages, not AIS, is (are) responsible for the accuracy of their content.
4. The author(s) of this article, not AIS, is (are) responsible for the accuracy of the URL and version information.

Judenberg, J. (1994) "Applications Maintenance Outsourcing: An Alternative to Total Outsourcing," *Information Systems Management* (11)4, pp. 34–38.

Kaplan, J., and J. Sikes (2009) "Managing IT Spending," *McKinsey Quarterly,* Issue 1, pp. 64–65.

Kyte, A. (August 2010) "How CIOs Can Maximize Value from Application Maintenance Teams," *Gartner Group*, ID# G00205509.

Kyte, A. (May 2011) "Estimating the Future Cost of Application Maintenance," *Gartner Group*, ID# G00212527.

McKeen, J.D., and H.A. Smith (2010) "Application Portfolio Management," *Communications of the Association for Information Systems* (26), Article 9, pp. 157–170.

Smith, H.A., and J.D. McKeen (2010) "Investment Spending Optimization at BMO Financial Group," *MISQ Executive* (9)2, pp. 65–81.

Swanton, B., and A. Kyte (2010) "How to Manage the Liabilities of Deferred Application Maintenance," *Gartner Group*, June, ID# G00200735.

Watson, R., et al. (2010) "Telematics at UPS: En Route to Energy Informatics," *MIS Quarterly Executive* (9)1, pp. 1–11.

Zensar Technologies Ltd. (2011) "Application Support & Maintenance," http://www.zensar.com/sofware-service/support-maintenance, July (current Nov. 2010).

ABOUT THE AUTHORS

James D. McKeen is a professor of IT Strategy and Distinguished Research Fellow in MIS at the School of Business, Queen's University at Kingston, Canada. Jim received his Ph.D. in Business Administration from the University of Minnesota. He has been working in the IT field for many years as a practitioner, researcher, and consultant, and is a frequent speaker at business and academic conferences. Dr. McKeen co-facilitates the networking of senior executives in the IT sector through two well-known industry forums: the IT Management Forum and the CIO Brief. In October 2011, Dr. McKeen was named IT Educator of the Year by IT World Canada. He also has extensive international experience, having taught at universities in the U.K., France, Germany, and the U.S. His research has been widely published in various journals, including *MIS Quarterly, Knowledge Management Research and Practice, Journal of Information Technology Management, Communications of the Association for Information Systems, MIS Quarterly Executive, Journal of Systems and Software, International Journal of Management Reviews, Information and Management, Communications of the ACM, Computers and Education, OMEGA, Canadian Journal of Administrative Sciences, Journal of MIS, KM Review, Journal of Information Science and Technology,* and *Database.* Jim is a co-author of three books on IT management with Heather Smith, the most recent being *IT Strategy: Issues and Practices* (2nd Edition) [Pearson Prentice Hall, 2012]. He currently serves on a number of editorial boards.

Heather A. Smith has been named North America's most published researcher on IT and knowledge management issues. A senior research associate with Queen's University School of Business at Kingston, Canada, she is the co-author of five books: *IT Strategy: Issues and Practices, IT Strategy in Action, Management Challenges in IS: Successful Strategies and Appropriate Action, Making IT Happen: Critical Issues in IT Management,* and *Information Technology and Organizational Transformation: Solving the Management Puzzle.* A former senior IT manager, she is currently co-director of the IT Management Forum and the CIO Brief, which facilitate inter-organizational learning among senior IT executives. She is also a senior research associate with the Society for Information Management's Advanced Practices Council. In addition, she consults, presents, and collaborates with organizations worldwide, including British Petroleum, TD Bank, Canada Post, École des Hautes Études Commerciales, the OPP, and Boston University. Her research is published in a variety of journals and books, including *MIT Sloan Management Review, Communications of the Association for Information Systems, Knowledge Management Research and Practice, Journal of Information Systems and Technology, Journal of Information Technology Management, Information and Management, Database, CIO Canada,* and the *CIO Governments Review.* She is also a member of the editorial board of MISQ-E.

Communications of the Association for Information Systems

ISSN: 1529-3181