THE DESIGN AND IMPLEMENTATION OF AN ACCOUNTING INFORMATION SYSTEM: WAYS TO ENSURE FAILURE

John E. McEldowney, Department of Accounting and Finance, University of North Florida, 1 UNF Drive, Jacksonville, FL 32224, 904-620-1104, jmceldow@unf.edu
Marcelle L. McEldowney, Time/Warner Customer Service, 1 North Dale Mabry, Tampa FL, 33609, marcelle.mceldowney@custserv.com
Michael P. McEldowney, GTE Data Services, 1 E Telecom Pkwy, Tampa, FL, 33637, Mike.McEldowney@tampabay.rr.com

ABSTRACT

Effectively designing even the simplest of systems can be a complex and daunting undertaking when political and personal agendas influence the process. If the procedural steps for controlling the design and implementation stages of a project are well organized and developed systematically, creation of a new information system or significant enhancement of an existing system can proceed smoothly and enable the proposed system to accomplish the goals for which it was intended. Unfortunately, in practice, the design and implementation of an accounting information system is oftentimes less than text book perfect. This paper will address a number of common design and procedural shortcomings which spell disaster for the development and implementation of any accounting information system.

INTRODUCTION

In today's environment, constant change seems to be the norm. This is especially true of information systems. With the advent of new technology and the increasing demands for faster and more accurate data for decisions, it is no wonder that managers are oftentimes faced with inadequate systems which no longer provide the information needed to compete in an increasingly global economy.

The logical solution to this dilemma is to create a process which continuously updates older accounting information systems with new technology for data processing, manipulation and dissemination. Obviously, any change to a current system should be well thought out, assessed for cost/ benefit, and carefully designed and implemented to ensure the minimum amount of disruption possible to current operations. And to help ensure smooth transitions from older systems to newly developed ones, a concerted effort must be made to approach each new systems development project with a structured approach.

Sadly, this is not the case in many systems development projects. Personal interests, poor planning, inadequate analysis, and improper implementation combine to create either systems that do not provide the benefits initially promised or simply do not function at all. While academic research has focused on a number of theoretical factors which can cause the failure of the systems development process, little has been written from the perspective of the developers themselves, from the analysts and programmers who are tasked with the responsibility for developing systems in the face of a seemingly endless number of conflicting requirements, personal agendas, and illogical demands.

This paper will address a number of reasons why systems development projects fail. From a practitioner's perspective, a set of factors will be identified that help to explain why these failures occur so frequently.

CREEPING ELEGANCE

The first factor that can significantly impact the development of any information system is *Creeping Elegance*. This factor addresses the complications that can arise when changes are made to the original user specifications for a project. In the initial stages of systems development, users normally specify their needs in terms of defining what the new system is intended to do, how the new system will behave in terms of the human interface, and what will be the projected operating speed for the components integral to the project. Unfortunately, the final project as initially envisioned by both users and IT rarely coincides with the original requirements. Oftentimes it bears no resemblance to what both parties agreed to at the outset of the proposed project.

The reason it is so difficult to establish a definitive view of the final project is because changes to the established requirements will inevitably occur. This change process of *creeping elegance*, can be divided into two categories: *type I* and *type II* changes.

A *type I* change refers to adjustments in the level of detail originally defined in the requirements phase. These are important changes that were overlooked but should logically have been included in the design of the proposed system. In essence, they were not sufficiently detailed by the user during the initial planning phases and were overlooked by IT. For this reason, *type I* changes are normally mandatory changes that need to be completed regardless of the cost, or potential impact on the deliverables schedule.

A *type II* change also represents features and functions that are added to the system after the requirements phase is complete. However, unlike *type I* changes, these new additions were never addressed by the user, nor were they logical extensions of the provisions outlined during the requirements phase. Left unchecked, these kinds of changes can represent a never-ending stream of alterations based on the user's newly conceived needs for the project under development. They are insidious, and allowing even one *type II* change to be added to the project's overall structure sets precedence for others to follow.

For any *type II* change, an impact statement needs to be prepared which specifies their estimated costs and project consequences. The fate of each of these *type II* enhancements should be negotiated on a separate basis, and should never be combined with the existing development package without first reviewing costs and scheduled impacts for the entire project. Left uncontrolled, *type II* changes become self-perpetuating. The more they are allowed to be added to the development process after the requirements phase is completed, the more *type II* change requests will be submitted. This will continue until they eventually overwhelm the project schedule and cost budget.

TECHNOLOGY FOR TECHNOLOGY'S SAKE

Another potential roadblock to the successful implementation of a systems project is selecting the newest technology simply because it *is* the most currently available technology. Rather than selecting software and/or hardware based on a careful cost-benefit analysis, these components are more often selected because a new IT product has been marketed well. A key question which is often overlooked is whether the proposed technology is even necessary to perform the proposed function. In essence, companies have a tendency to jump into new technologies without a proper business justification. The lure of state-of-the-art technology is often too hard to resist.

The "real world" is replete with examples of this kind of mentality. For example, one large corporation was pressured into transforming its system from a virtual sequence access method (VSAM) to relational database platform (DB2) by a well-meaning, but unfortunately, technologically illiterate group of upper level managers. They felt that using the most technologically advanced software would transform their "antiquated" system into one that would provide a panacea for all their data processing issues.

Unfortunately, their efforts resulted in additional costs of overhead, support and maintenance due to the complexity of the new system and, ironically, it carried with it virtually no return for these costs. The updated system was slower, more complex, required more maintenance costs and resulted in additional run-time that was not anticipated. The VSAM they currently had in place was able to provide them with the information they needed because the relational database functions they acquired with the new DB2 system were never needed in the first place. The firm had adopted technology for technology's sake. New systems were put into place without an adequate analysis of actual needs and costs.

MAINTENANCE PHASE REVELATIONS

Maintaining a newly developed system can also present significant roadblocks to systems development. Any system that is implemented will ultimately require changes to that system due to advances in new technology, inadequate processing capabilities, or changes in a firm's information needs. For most systems, these changes are controlled and processed during the last stage of the systems development life cycle – the maintenance phase. This phase also tends to be the most expensive portion of the development process. Between 70% and 80% of the *total* costs for system projects are incurred in this phase. Once a system has been analyzed, designed and implemented, the firm has incurred only 20% - 30% of the total cost it will incur with respect to this system. The remaining portion constitutes the costs associated with maintaining the new system. This is an important aspect of any system proposal, and one that is oftentimes overlooked or downplayed.

Any company can build an elaborate system, but, because of the system's design or the experience level of the staff, it may not be maintainable. For example, a service order system was developed for a firm where the application code for the system existed on each of the functional terminals utilized in this system. This enabled each terminal to function as an independent processing station, but little thought was given to the possible repercussions of this action.

Because of the design of the new system, whenever a problem occurred on a particular workstation, the standard procedure was to change the application code on that terminal to rectify the problem. Unfortunately, the ramification of this procedure was that when the code was updated on one terminal, 49,999 other workstations needed to be upgraded with the same source code changes as well.

Additionally, there was no established method of propagating that source code repair to every other workstation. If there is only one copy of source code that is used by numerous terminals, the updating process is relatively simple; merely update the copy that all terminals access. In this scenario, however, there were literally 50,000 copies of the source code spread throughout the U.S. and Canada with each workstation being allocated its own individual copy of that code. The firm had no single source for the application code and therefore ultimately had no way of disseminating any individual changes in the code to each service workstation. This is a prime example of failure in the design of the maintenance phase of an application, because no consideration was given to how laborious and time-consuming changes and updates can be to an implemented system.

MANAGEMENT STYLE MISMATCHES

In the systems development process, there are discrete phases of a project that must be completed in a logically sequenced order. Each of these phases must be managed appropriately or "runaway" projects in terms of excessive costs and delayed implementation dates can ensue. Placing the wrong management type in one of these stages can significantly hamper the systems development effort.

For example, personnel that should be used in the requirements and high-level design phases are those who can think "outside the box". These types of managers should be creative freethinkers. They are

often viewed as having a "blue sky" type of personality because they view issues from a broader perspective, create more innovative solutions and thrive in environments where "the sky's the limit."
For the detailed design and coding phases, individuals should be more task-oriented. Ideally they should have structured personalities that demand a detailed, hour by hour project management plan to follow. Everything they do should be oriented towards meticulously following the guidelines established for completing projects. For this type of position, the person doing the work should *not* be someone who is good at thinking outside the box. Here the personality needed is one like the stereotypical bookkeeper; the personality type is someone who won't get bored with the details.
For the Implementation and Maintenance phase the perfect personality type would be that of a firefighter. The person who will excel in this area, and the person most needed to be placed in this part of the process would be one who thrives on stress. He or she works best where no two identical things occur sequentially. Here a person with multitasking skills is imperative. The individual needs to be able to react quickly and effectively to emergencies, because in this phase, these conditions are common. He needs to be able to juggle activities and prioritize them in the best order for the project.
Now the problem is, if the wrong person is placed in a key position, it can create a nightmare for a systems project. For example, placing a *blue sky* person into a unit test environment will ensure that these tests will never be completed. Positioning a person who is a free thinker in a mundane, repetitive area like detailed coding, ensures the project will be late, over budget, or may never be completed.
Mismatching management styles is commonplace. Individuals are continually being placed into positions where they are not well suited. The key to overcoming this potential roadblock is to devote more time to assigning individuals to individual parts of a project. Strengths and weaknesses of employees should be assessed more candidly with the goal of fitting the right person into the right position. Additionally, a more systematic approach is needed in yearly evaluations to identify personnel who are more suited to particular functions.

INTRICACIES OF THE MR. FEEL GOOD ROLE

The "Mr. Feel Good role" is not actually a role connected with any particular individual in a systems development project. It is instead, a composite of multiple personnel and their reporting responsibilities. Mr. Feel Good is a fictitious person who ensures that everyone from executive sponsor to the lowest level supervisor is kept abreast of what is presently occurring in the project. This role is viewed as fictitious because no single person does all of this. Different people have different responsibilities that all contribute to good project communication. So, the Mr. Feel Good role is synonymous with the communications function of a systems development process..
Miscommunication can cause a serious user involvement problem. From a control and follow-up perspective, good communication helps ensure that individuals cannot claim they were unaware of important issues that developed during a project. This helps prevent situations where individuals maintain that since they weren't informed of a particular action, they cannot support it. Failure to communicate well in the development process creates a serious roadblock where individuals or departments refuse to participate in project because they were not notified properly and were therefore unable to prepare accordingly. Too often in large system development ventures, communication either does not occur, or not at the right level. Controls must be put into place to help ensure that everyone is kept abreast of the status of a project at any one point in time.
Important communication devices that should be employed entail the use of issue lists and change lists. Issue lists are regular alerts notifying interested parties about setbacks or roadblocks in the systems project. Weekly meetings should be established to discuss these issue lists. Key personnel from departments impacted by items on the list should attend these meetings and appropriate responsibilities and due dates for resolution should be assigned.

Another communication mechanism that can support a positive Mr. Feel Good role is a requirements change list. This listing addresses whether there is enough time, money, and justification to make a particular change. If a user wants to propose a new change in the development process, a requirements change list should be created and distributed among the affected parties. This listing should specify what impact the proposed change will have on the work plan, the budget, and the project time line. When requests are approved, then all parties affected by these changes should be notified of the adjustments mandated. Additionally, the potential impact of these changes on the development schedule as well as newly assigned responsibilities for implementing those changes should also be communicated. Whatever the issue, good communication via Mr. Feel Good is essential for the success of any systems development process. Changes occur; they are inevitable. How timely, efficiently and accurately they are responded to oftentimes determines the success or failure of a project.

ELUSIVE REQUIREMENTS

In a normal IT environment, if the client was allowed to, he would want his house/project built, he would want it *yesterday*, and he will provide specifications, like square footage and the number of rooms, *next* week. This is commonplace because creating specifications is a difficult, time-consuming process, and one that users would gladly concede to the IT function. There is a natural inertia on the part of users for developing requirements for a new system. Sometimes the root cause is that they are not sure what it is they really want, and other times they are unsure of the options available to them. Regardless of the underlying reasons for their lack of specificity, defining what is needed is an arduous process for most users, and avoidance is the logical recourse.

A compounding problem related to elusive requirements is that users oftentimes want things done in a manner that is "comfortable" for them. Even if the accounting information system is being converted from a manual to a computerized system, users may want to see the exact same format for the processes they are familiar with. Instead of starting with a high level design that addresses the needs and key objectives of proposed systems, users oftentimes press more for compatibility with the previous system that is being replaced. Rather than focusing on what the new system will achieve to meet the users' needs, user groups fixate on minimizing the changes to the way things were done before. They frequently have their "comfort zones" established and do not want them to be modified to any real extent, regardless of the problems they are experiencing with their present system.

Because users, will normally not commit the time to write detailed requirements for a proposed system, IT must take the initiative in soliciting their involvement. The best way to address this problem is to establish a steering committee composed of IT personnel and representatives from user groups. Lines of communication need to be opened between the two groups and responsibilities of each group outlined in detail. IT cannot afford to allow users to ignore their obligations. At best, doing so invariably leads to having to prove that what was finally delivered to the user is what was asked for in the first place. At worst, resources will be expended on a new system that does not meet the needs of the users and therefore will be not be used by them.

Better educating users as to how the process of developing a new system will proceed will also help minimize the chance of project failure. Users often assume that if they write down their specifications in detail, reality will follow their wishes. Unfortunately, this is not how things happen in the real world. Users must be led to understand that any plan of action must take into consideration three factors: scope of the project, resource constraints, and risk minimization. In essence, the development team must balance the scope and quality of a proposed project against time, available resource constraints and the necessity of minimizing the risk that the system will fail. For example, if the scope of a project is unrealistically broad, it will be nearly impossible for the resource, time and quality constraints to result in a finished system. Therefore, the risk of failure under these conditions would be extremely high.

There are few options available to save a project that was inadequately planned. The only alternatives would be to reduce the scope of the project, increase the available resources to complete the project, or reduce the quality of the finished product. These factors of scope, risk, resources and quality must be balanced for the successful completion of any systems project.

Another tactic that can be used to solicit more user involvement is to review and determine approximate costs for the items they initially want in their new system. When costs are directly associated with the discrete components of a proposed project, the user is less likely to submit a "wish list". At this point, most users realize their error in asking for too much. They can't afford it and instead need a more realistic scope for the proposed project. If users perceive IT services as "free" and without constraints, most of their requirements requests will be unrealistic.

CONCLUSIONS

Effectively developing even the simplest of systems can be a complex and daunting undertaking when political and personal agendas influence the process. If the procedural steps for controlling the design and implementation stages of a project are well organized and developed systematically, creation of a new information system or significant enhancement of an existing system can proceed smoothly and enable the proposed system to accomplish the goals for which it was intended. Unfortunately, in practice, the development of an accounting information system is oftentimes less than text book perfect. In the “real world”, a surprisingly large number of systems projects are unsuccessful due to various factors which can cripple the development process. These issues can make the successful implementation of an information system untenable at best.