

# Top 10 Reasons Why Systems Projects Fail

*Learn from an experienced project manager how to avoid the common pitfalls that often lead to the failure of information systems projects.*

by Dr. Paul Dorsey (pdorsey@dulcian.com)

## Overview

Information systems projects often fail. Depending upon what statistics you look at, the failure rate of large projects can be 50%-80%. Since few people like to admit failure, the real statistic may be even higher. This is a catastrophe. As an industry we are failing at our jobs.

As the term indicates, software “engineering” is truly a kind of engineering. Building a large information system is like constructing a 20-story office building. If a bunch of electricians, plumbers, carpenters and contractors meet in a field, talk for a few hours and then start building, the building will be unstable if it even gets built at all. At one of my presentations, an audience member shared the quip that “If building engineers built buildings with the same care as software engineers build systems, the first woodpecker to come along would be the end of civilization as we know it.”

How can we avoid making the mistakes that lead to project failure? Surprisingly, the answer is most often simple common sense. Our problem is that common sense is often ignored in systems projects.

What causes so many systems projects to fail? Is it some technical secret that most systems engineers don't know? Are software projects so drastically complex that only the most talented teams have a hope of succeeding? The answer is that software projects fail because we do not recognize that good engineering principles should be applied to software projects just as they are to building office buildings.

## **Three Keys to Successful Projects**

There are three factors that all successful projects seem to have in common. Each project can be viewed as a tripod. All three legs must be in place for the tripod to stand sturdily:

- Top management support
- A sound methodology
- Solid technical leadership by someone who has successfully completed a similar project

### **A. Top Management Support**

Almost all studies of system success or failure have identified top management support as a critical success factor. The management personnel in any organization that undertakes a systems project should be aware up-front that the project will encounter serious setbacks. They will need to be prepared to remain visibly and vocally behind the project, despite these setbacks or else the project is doomed to failure.

There is a real difference between systems projects and office buildings. When a building is half done, there is something to see. When a software project is half done, there is very little (if anything) to see. Managers need to know what they can expect to see and when. If they assume that the project will have 50% of the systems running when the budget is 50% spent, they will probably start thinking about pulling the plug on a project that is progressing exactly on schedule.

Managers often do not understand the design of a system. They rely on the opinions of skilled advisors. The key to managing the managers is to bring in high-level objective auditors. In a consulting environment, this is particularly important. How can management know that they are not being cheated or that the project is not being mismanaged? They don't have the skills to assess the situation. In such cases, having a technical audit can validate the actions of the development team and provide management with the information required to continue supporting the project.

## **B. Sound Development Methodology**

Many systems are built with little thought to process. The team gets together and starts performing activities. As soon as enough information is gathered, coding begins. This lack of attention to process can destroy a system. It is easy to see the result of a lack of attention to process after a system fails. Usually major portions of the user requirements are ignored. Large amounts of code need to be rewritten, since it does not meet user requirements the first time around. If completed, the system is put into place with inadequate testing. Without a well thought out process, there is little chance that a systems project will be completed. If the project does succeed, it only does so with substantial rewrites and cost overruns.

It may be surprising to think that the methodology selected doesn't matter, but in fact this is basically true. What does matter is that there is SOME methodology. There is no explicit research to indicate that any one particular methodology is better than any other. Different methodologies all gather the same information but organize it differently. One may have additional, unnecessary steps. Another may miss some steps requiring developers to backtrack to earlier phases. It can be useful if the methodology selected is tightly integrated with the development tools selected since there will be less wasted effort. However, this still will not guarantee project success. A project may be more or less expensive but won't succeed or fail based upon the methodology or tools used.

## **C. Technical Leadership**

Just as a building needs an architect, so a software system needs a technical lead. The technical lead must have built similar systems down to the level of the specific business area for which the system is being built.

To be successful, the architect or technical lead must be the one in control of the "architecture" of the project, namely the data model and application design. This level of control must be recognized and acknowledged by everyone involved with the project.

## **Interdependent Factors in Project Success**

In any systems project, there are four interdependent factors:

- Cost
- Quality
- Speed
- Risk

It is not possible to have the best of all four factors. Specifically, you cannot have a system built inexpensively, of high quality, built quickly and with little or no risk of failure. Most discussions of these factors only include the first three. It is possible to build a high-quality system quickly, at a relatively low cost by cutting corners, and doing little or no testing. However, the risk of such a system failing increases dramatically. Of these four factors, in any project, two are always possible to achieve successfully, leaving the other two to be managed.

Of these four factors, the two most important are risk and quality. The system must work and successfully meet user requirements. This leaves speed (time) and cost (money) to be adjusted accordingly. If you insist on fast development time or low cost, then quality and risk will shift accordingly.

## **Data Migration and Implementation**

Two additional factors in determining the success or failure of a project that are often forgotten are data migration and the system implementation itself. Data Migration should be planned for early on in any project. Data Migration should almost be considered as a separate project in his own right.

Similarly, even a well-crafted, well-documented and carefully designed system can still fail 10-20% of the time because the implementation is not handled correctly. This can be due to inadequate training of users, poor transitioning from the old to the new system and lack of user support for the new system.

## **Top 10 Ways to Guarantee the Failure of a Systems Project**

The following list has been inspired by actual mistakes encountered in real-world systems projects.

### **1. Don't use a specific methodology because coding is all that is really important.**

Using a structured systems development methodology is one of the critical success factors in a systems development project. For this reason, some years ago, Peter Koletzke and I came up with the CASE Application Development Method (CADM) as a successor to the traditional CASE method pioneered by Richard Barker. CADM, documented in the Oracle Press book *Oracle Designer Handbook* (Koletzke & Dorsey, 2<sup>nd</sup> Edition, 1999) is based upon a few core concepts. First, an engineering manufacturing approach to software development was used. This approach pays careful attention to quality control points, identifying where in the Software Development Life Cycle (SDLC) these points occur and what types of checks are necessary to ensure that each phase is successfully completed before the project is ready to move into the next phase.

A second major philosophical component of CADM is an audit trail for system requirements. We included the ability to track a requirement throughout the SDLC from its initial gathering during analysis to its impact on user acceptance testing at the end of the process.

### **2. Create the project plan by working backwards from a drop-dead system completion date.**

Working backwards from a fixed project completion date is a sure way to guarantee project failure and yet, unfortunately, this is an all too common practice in the IT community. Managers may make a decision about when a new or re-engineered system would be useful to have in production without the necessary technical knowledge to determine whether or not it is possible to accomplish successfully in the given time period. No project ever went completely smoothly from Strategy to Implementation. There are many places in the SDLC where schedules can go awry:

- Failure to perform careful analysis resulting in critical new requirements being uncovered late in the development process
- Failure to take data migration into account

- Failure to accurately assess the political climate of the organization vis à vis the project
- Failure to enlist approval at all levels of the user community

You must set a realistic timetable for the completion of a systems project and expect that some deadlines will slip as unexpected setbacks inevitably arise.

### **3. Don't bother with a data model. Just build whatever tables you need.**

The data model is the core of the system. Without a carefully constructed data model, any system is doomed to failure, or at least to not meeting the users' needs and requirements.

Data models should be directly under the control of the technical lead. Each subsystem should be integrated into the whole prior to development.

Models should be designed and audited, and re-audited, and re-audited until no more errors are uncovered. The audits should first be done by the technical lead. Then an outside auditor should always be brought in for an additional audit.

### **4. Use a Technical Lead that has never built a similar system. Hiring such talent is too expensive.**

The person in the role of project Technical Lead must be experienced. He/she should have completed other successful similar projects. The role of technical lead is like that of a skilled heart surgeon. You would not expect a family doctor to perform brain surgery or administer anesthesia. It is critical for the person in charge of the technical aspects of the project to have experience with the type of system being built. Of course, such a resource is quite expensive, but not nearly as expensive as a failed project.

### **5. Hire forty developers to make the coding go faster.**

More is not always better. This is especially true in the case of development teams. A project with a few skilled developers, carefully supervised by the appropriate technical lead has a much greater chance of success than one where hordes of developers are cranking out mountains of code each day.

**6. Build the system in Java, even though most of the development team still thinks that Java is coffee and you have no intention of ever deploying to the Web.**

“The right tool for the right job.” This motto is as true for building systems as it is for building houses. One of the problems for many system architects is the fact that the available tools greatly influence the way systems are built. Even the basics of the development process are influenced by the tools selected. Make sure that the necessary expertise is in place for whatever development tool is selected.

This push to Java and the Web will certainly give rise to a whole new category of system failures. In case no one has noticed, the Java environment is still relatively immature. We all still have a lot to learn about Java and web development.

**7. Three months before the system goes live, assign one junior developer to handle the data migration.**

The data migration phase of a project can consume up to 30% of the total project resources. Typically, the amount of attention paid to it in existing methodologies is very small. The cost of data migration tends to go unforeseen, until far too late in the project schedule. The most common flaw in data migration planning is that too few resources are invested in it.

For these reasons, it is very important to plan ahead for data migration on any project. In particular, implementing generic data structures, which are desirable for increasing the lifetime of the system, can create significantly more complex migration issues than implementing traditional system designs.

**8. Skip the testing phase because the project is way behind schedule.**

Putting a system into production without adequate testing is like diving into a swimming pool without checking to see if there is water in it. No system was ever created completely bug-free. The time spent to thoroughly test any system before placing it into production can save much more time in the long run.

In the Test phase, the lead QA person needs to develop a test plan that should describe not only the tests to be run but also how test failures or variances will be handled. There are two components to a test plan: the approach and the design.

The approach includes the testing techniques, testing tools, roles and responsibilities, method of error detection and logging, change control process, re-testing process, and testing environment. The design part of the test plan includes the test objectives, cases, and scripts. Users need to be involved to identify the test cases, and they can write the test scripts and expected outcomes.

It is not necessarily true that every test failure or variance leads to modifications of the system prior to production. Within the Test phase, it will be necessary to re-audit the design process, perform system- and user-level tests, audit the quality of the documentation, test the internal controls and backup and recovery procedures, and in all ways ascertain the fitness of the system to move into a production environment.

## **9. Change the system to support critical new requirements discovered during final development.**

It is important to prevent the onset of “scope creep” on a project. There will always be new requirements to support, slightly better ways of modifying your data model and better ways to design an application. Unless you set a formal rule that prevents the majority of well meaning suggestions about improvements from “creeping” into the system, the system will never be completed.

Who decides what will get done and when? Throughout the system life cycle, there should be a management team made up of the project leaders, a representative of upper-level management, one or more functional area users and, perhaps, a DBA (if no one else on the team has DBA experience). This small management team controls the CADM process and either performs all of the review tasks or delegates them to appropriate individuals.

As you progress through each phase of the project, you have more and more to worry about with regard to changes to earlier phases. In the Analysis phase, you only have to worry about changes to the Strategy phase; whereas in the Design phase, you have to worry about changes to system requirements and changes in scope from the Strategy phase.

Systems are not like works of art that, once completed, are hung in a gallery. They are inherently evolving structures that will inevitably need enhancements and subsequent versions. Every requirement does not have to be included in Version 1. It is much better to have a system that is adequate and running than a perfect system that is never delivered.

## **10. Buy a commercial, off-the-shelf package and customize it ... a lot.**

The only successful way for a commercial off-the-shelf (COTS) implementation to be successful is to decide at the outset that you are going to reengineer your business to fit the limitations of the COTS package. These packages usually imply a specific method of doing business with a corresponding set of capabilities. A managerial decision must be made that whatever the package does is good enough and that customization is not an option.

If you do not follow this philosophy, implementing a COTS package is no cheaper and no less risky than building a system from scratch. I have heard more horror stories with respect to implementations of large, famous ERPs than any other type of systems project.

### **Conclusions**

In order to ensure system success, there are several factors to keep in mind:

- Don't cut corners, methodologically. In the long run, this results in system failure or an inadequate system that doesn't meet the users' needs.
- Audit each major deliverable and step along the way for accuracy and correctness.
- Carefully monitor top management support for the project. Make sure that managers are aware of the progress of the team.
- Secure the correct technical lead for the project.

There is no free lunch in software engineering. If you insist on keeping costs low and hurrying the project along, then quality will be low or the risk of failure will be high no matter how well the project is managed.

## About the Author

Dr. Paul Dorsey (pdorsey@dulcian.com) is the founder and President of Dulcian, Inc., (www.dulcian.com) a New Jersey based Oracle consulting firm that specializes in data warehousing, web and client-server systems development and products that support the Oracle environment. Paul is co-author with Peter Koletzke of Oracle Press' *Oracle Designer Handbook (1999)*, *Oracle Developer Forms and Reports: Advanced Techniques and Development Standards (2000)* and with Joseph Hudicka of Oracle Press' *Oracle8 Design Using UML Object Modeling (1999)*. Paul is an Associate Editor of SELECT Magazine and is President of the NY Oracle Users' Group.