Top 10 Reasons Why Systems Projects Fail

Dr. Paul Dorsey
Dulcian, Inc.

Overview

Information systems projects frequently fail. Depending upon which academic study you read, the failure rate of large projects is reported as being between 50%-80%. Because of the natural human tendency to hide bad news, the real statistic may be even higher. This is a catastrophe. As an industry we are failing at our jobs.

Based upon past experience, when undertaking a large, complex, systems project, the realistic expectation should be that the project will fail. In particular, business process reengineering (BPR) projects have an even higher failure rate because of their expanded scope. Hiring a large, established consulting company is no guarantee of success; neither is buying packaged software and implementing it.

Projects are frequently built using a strategy that almost guarantees failure. Software engineering is 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."

If we would keep in mind that building software is just like building an office building, then most of our problems would go away. Before building an office building, an architect draws up detailed plans, builds models, and makes blueprints. At each stage, the plans are carefully and repeatedly reviewed. During construction, each step is tested and inspected. In software projects, if we haven't started writing code in three months, we mistakenly think that something must be wrong. Can't we see the senselessness of plunging ahead without a careful plan? As an example, on one project I was involved with, the project manager brought in the tech writing team to start writing the training manuals before the user interface was even stable. This is about as smart as bringing in the painters before the walls have been built.

So, what really causes these 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? I have personally been involved in both successful projects and projects that crashed and burned. The sad fact 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. We try to defend ourselves by saying that software construction is "different".

When Peter Koletzke and I first published *The Designer/2000 Handbook*, we sneaked in quite a lot of material on software engineering. We tried to write about how one should best build a system using the Designer product. The way that this material was received by the community is interesting and sometimes negative. One reader quipped, "I already know how to build systems. All I need is to know what to click on in the tool." Another complained that there were no insights in the book and it that

was just a rehash of the "SDLC junk that they made me read in school." More experienced systems engineers that had been involved in numerous projects saw the value of the book and were quite receptive. But I think the most interesting comment on the book came from my brother Walt. Walt is a "real" engineer who has managed the quality function for several large technical manufacturers. He told me that he liked the book and that the advice in the book largely mirrored the way that large projects are managed in the companies he worked in. He was surprised that this kind of development method was not part of the normal way that systems are built.

I have given many presentations at Oracle conferences. If I talk about how to improve database design, I can usually attract 30-50 listeners. If I talk about improving the Java development environment, I can get 300-500 listeners. I believe that this lack of interest in methodology is at the heart of our problems. Many of us mistakenly believe that the only real contribution to a project is writing code. Using the office building example again, this is like saying that if you don't have a hammer or paint brush in your hand, your contribution to the building is not significant.

Don't expect to read any brilliant insights in this paper. I will share with you my successes, failures and my near catastrophes. My ultimate conclusion is that the best way to make a project succeed is to use your head. 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.

The 3 Keys to Project Success

There do seem to be three factors that all successful projects have in common. Each of these factors is key to any project's success. Each project can be viewed as a tripod. All three legs must be in place for the tripod to stand sturdily. In a systems project, these "legs" or critical success factors consist of the following:

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

Without each of these solidly in place, the tripod will topple and the project will fail.

Top Management Support

Every study ever done about system success or failure has identified top management support as a critical success factor. Without full commitment from top management, when problems arise on a project (as they inevitably do), the project will collapse. 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.

I was involved in one system where the implementation was going poorly. Rank and file users were about to revolt. However, top management stayed behind the project and it eventually succeeded. If management had not been as committed, the project would surely have failed.

In working on another system, the project was going fine, but a new manager came in and the project just disappeared overnight. In fact, a few of the failed projects that I have been involved with were canceled by a person whom no one on the development team had even met. This leads to an important point that a project will not be successful if management doesn't think it is successful. Management needs to be educated about the process being used and their expectations must be managed.

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.

Beware of the skilled developer who thinks he/she is a project lead. An experienced developer with several years experience may not understand the design of the system. However, they can single-handedly kill a project with a well-placed opinion.

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. A project can be ended by management simply because they misunderstand the actions of the development team. 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.

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 kill 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. What is important is keeping the project organized in some consistent and focused way and thinking through the process carefully at the outset.

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. The important point is to use some methodology successfully. If a process is flawed, usually it is not seriously flawed. One methodology may be more efficient than another, but any process is better than no process.

There are many ways to approach systems development – object-oriented, rapid prototyping, waterfall, etc. They each use different tools to accomplish the same tasks. An object-oriented approach would employ use cases in analysis, whereas a traditional Oracle professional will use a function hierarchy. If a particular organization has talent in one specific area, there is no reason not to exploit that expertise. For example, if you have a number of object-oriented developers, an object-oriented approach would be a clear choice. Similarly, a shop with Oracle Designer expertise might use the CADM methodology. As I have stated elsewhere, 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.

Technical Leadership

Just as a building needs an architect, so a software system needs a technical lead. 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. Otherwise, each portion of the system may be constructed independently by a portion of the team and the pieces won't fit together at the end.

The technical lead must have built similar systems down to the level of the specific business area for which the system is being built. For example, any system that includes financial functions must usually interface with existing accounting functionality. This means that the technical lead must understand basic accounting practices. In my own experience, I have been asked to review failed projects including financial functions where previous system designers and tech leads did not understand the basics of debits and credits.

Interdependent Factors in Project Success

In any systems project, there are four interdependent factors:

1. 1. Cost
2. 2. Quality
3. 3. Speed
4. 4. 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. I have had many arguments with managers over this principle. I have been told "If we use Product X and Methodology Y, we can have the system built quickly at minimal cost." This is not a realistic or tenable position. You can insist on low risk and high quality, recognizing that time and money must be adjusted to achieve these goals.

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.

List of the 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 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. Explicit quality control points must be included in any successful project. For example, the formal model defense after the data model is complete is an often skipped, and later regretted, step. Conversely, within CADM, quality control points were eliminated where unnecessary due to implicit controls. For example, performing data migration helps to validate many aspects of the physical database design.

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. Forcing a deadline to drive the schedule will cause corners to be cut, greatly increasing the risk of the project. Even if the system gets built, it is usually of very poor quality as application after application is built just to get them completed.

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.

I am always surprised when I hire a developer with several years of experience who has never seen a data model. Even if they have worked with data models, they have usually never seen anyone carefully audit one.

I am equally astounded when I go into a large financial institution and find that there is no data model for a system that has several hundred tables. When I audit such a system, I find that there are numerous critical flaws in the system that allow bad data to be entered. In such systems, data migration to a new system can take many months because so much of the legacy data has to be cleaned up.

The worst situation I have ever seen had each developer building whatever tables and applications they needed to support those tables. The technical lead came in later to try to make the disparate pieces work together. Needless to say, the system was never completed.

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 - not nearly as expensive as a failed project, but still expensive. I have to admit that I am mystified by managers that will gladly put 5-10 more $700/day consultants on a project but will refuse to accept that a good project lead will cost several times that amount. So, they have a large team generate a bad system for a great hourly rate.

My favorite example was a large organization with 80 developers that were unable to bring up a system. In fact, according to the project manager, little progress was being made on the system. However, the same organization was unwilling to bring in a senior technical lead that could have put a solid development environment in place, and would have made all 80 developers productive.

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.

On one system where I was hired as technical lead, the project had 40 people assigned to it. I was asked what resources I needed to make the project succeed. I told them "20 fewer people." They thought I was kidding, I wasn't. However, this was a consulting firm, where billing was evidently more important than efficiency.

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.

One client recently informed me of their intention to deploy web forms to their users. However, their user base consisted of 20 users all located in the same building using two networks. Such a system could easily be supported with a simple client server system for far less money. Other companies have set formal company policies that new development will be done in Java, independent of the cost of the decision or the skills of their developers.

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. Many project schedules will list data migration as a single task in a development effort. However, by project's end it will be evident that data migration was indeed an entirely separate project, highly dependent upon several of the deliverables of the development project.

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.

After witnessing a number of failed projects and learning from bitter experience, waiting three months before going production to assign one person to handle data migration will inevitably cause system failure.

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, you should develop a test plan that should describe not only the tests to be run but also how test failures or variances will be handled. Especially for a large system, it is not practicable to wait until the end of the Build phase to start developing this test plan. There must be some overlap. As each module is tuned and passes unit-level testing, it is stable enough so that formal tests can be planned.

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.

The lead QA person needs to develop a test plan. Users need to be involved to identify the test cases, and they can write the test scripts and expected outcomes. 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. The test plan is the framework in which the different levels or types of testing are performed, i.e. system testing and user acceptance testing. You should develop a test plan for each type of testing.

In one system that I worked on, the company insisted on doing testing by putting the system into production and letting the users find the bugs. When the bug filled system was put into production, the users were furious.

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. Once you freeze a deliverable at any phase in the CADM process, it should not be altered. The only exception to this rule is when a mistake in the design of the system is uncovered that will prevent the system from going into production. If you do not enforce hard and fast rules about what changes are acceptable and when changes can be made, the system can be delayed for weeks, months or never go into production at all.

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. The process of review and quality assurance is critical to the success of the CADM process and should be handled by the best talent available. Of course, a key principle is that no one can perform a QA review of his or her own work.

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. At each phase, we will identify all of the major deliverables from the earlier phases and discuss the impact of changes to any of those deliverables. Ideally, this information would be displayed in a giant matrix with all of the deliverables lined up on one axis and all of the phases on the other in order to clearly see the impact of changing each deliverable for each 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.

Just recently, I flew back on a plane with a manager who was recounting his horror story. His company is a $30,000,000 a year manufacturer. Two years ago they were sold a large well-known ERP. The project had a 2-year schedule and a two million dollar budget. After two years and four million dollars, the project is 10-15% complete.

Conclusions

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

1. 1. 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.
2. 2. Audit each major deliverable and step along the way for accuracy and correctness.
3. 3. Carefully monitor top management support for the project. Make sure that managers are aware of the progress of the team.
4. 4. 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) an 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, Oracle Developer Forms and Reports: Advanced Techniques and Development Standards* and with Joseph Hudicka of Oracle Press' *Oracle8 Design Using UML Object Modeling*. Paul is an Associate Editor of SELECT Magazine and is President of the NY Oracle Users' Group.