



The Menlo Institute LLC

# Project X

A Software Development Case Study

**Thomas Meloche © 2002  
Fellow, The Menlo Institute**

**The Menlo Institute LLC  
410 N. Fourth Ave, 3rd Floor  
Ann Arbor, MI 48104  
[www.menloinstitute.com](http://www.menloinstitute.com)  
(734) 665-1847**

## **1 Introduction**

This paper highlights how to improve software delivery management practices by examining a real-life project. This study is about a specific team's effort to deliver their first object-oriented project. With the advent of Java and the Internet it seems to be a particularly relevant case study for most organizations.

In the world of software development there are many examples of great failures and a few less examples of great successes. This paper reviews a project that stands somewhere in the middle. It is a muddy mixture of both failure and success where the software is late, over budget and buggy yet somehow actually delivered value to real customers. In fact, it was still in use over seven years after its initial release. It was just recently replaced by a second generation system - but that is another story. For privacy reasons key identification information has been changed to protect the innocent and the guilty.

### **1.1 Project X**

The software project we will review has been code named Project X. The code name is appropriate because the project involves building an image management system for X-Ray Departments. Project X was conceived with the best of intentions: to improve the quality and reduce the cost of medical care. The system was simple in concept. Its goal was to reduce the dependency on film in the X-Ray Department by replacing physical films with digitized images stored on a server. The images could then be viewed on personal computers that were connected to the server via a network.

### **1.2 Business Drivers**

Before we begin analyzing the development of Project X, it is advantageous to understand some of the reasons why such a system would be built. A digital x-ray system has several advantages over its film counterpart.

Advantage 1: Digitized images cost less than film. Film is an expensive medium. The cost of x-ray film production and management in a large hospital may well be over a million dollars a year. It is expected that a digital medical x-ray system will pay for itself in about 16 months.

Advantage 2: Digital film is harder to lose. Physical films are prone to be lost. In some institutions the long term loss rate for films may be 10% or more. Digital image loss rate is expected to be 1/10 that of traditional film.

Advantage 3: Diagnostic time can be greatly reduced, thus resulting in better patient care. Digital films can be transmitted over networks to

physicians in remote locations. The time it takes to provide a diagnosis for an x-ray can go from days or weeks to just hours.

These were simply a few of the many advantages customers of the software could achieve when moving to a digital medical record. It was a compelling system to build for the customers and it was hoped that the project would turn a tidy profit for the company developing it.

### **1.3 Company Y**

So, who was chartered to develop this amazing software for Project X? Company Y, who was an industry leader in X-Ray Machine Manufacturing. Although their medical hardware contained some rudimentary software, they were not a software development company by practice or culture. They saw the world through the hardware-manufacturing viewpoint of manufacturing schedules, assembly lines and shutdowns.

Company Y had never produced or sold software independent of their large medical imaging devices. In fact, they had a hard time believing that people would pay a premium for software that was independent of specialized hardware.

## **2 The Prototype**

Company Y hired several individuals who had built medical imaging systems in the past in order to overcome their inexperience in developing commercial software. In the process of hiring these individuals, Company Y also licensed some existing software that performed similar functionality.

Company Y decided that they would use the licensed software to build a fully functional prototype for Project X, in order to gain critical insights into this market.

Project X developers leveraged the licensed code to make a fully functional prototype of the x-ray image management system. The prototype was produced by a small team of four people over a period of four months. The software was not well designed or tested. It was not meant to be used in production. It was created to gain a better understanding of the market. The prototype, although extremely buggy, was well received by the test sites. The potential customers immediately saw the value and cost savings from a digital medical x-ray film. In fact, all of the prototype sites would become customers of the final product.

A final deliverable during the prototype stage was a draft version of a product definition and requirements document. This document captured the lessons learned from creating the prototype including feedback from the test sites. It also

## Project X: A Software Development Case Study

described the proposed new software that included enhancements based on this feedback. The document was not final and would continue to be updated throughout the entire development process with new features and requirements as they became known.

### 2.1 Project Commitment

Given the enthusiastic reception for the prototype, Company Y decided to commit to the production of a complete system. They added staff to the development team. The team now included:

- 1 Manager
- 6 Software Developers
- 1 Field Integrator
- 2 Software Testers

The 2 software testers were located at the main corporate headquarters 2,000 miles away from the software development team.

Company Y management decided that quick delivery was important for the new release. They wanted to ship the new software within ten months. Not waiting for the actual product to be started, much less completed, Marketing and Sales immediately began to sell the software based upon the success of the functional prototype.

### 2.2 Project Scope

Project X was not a small initiative. It involved the coordination of four major applications often running simultaneously over a distributed network. The four major components were as follows:

**X-Acquire** - The software component that digitized the images. It needed to work with the x-ray machine to acquire the images and then write them to the X-Store component.

**X-Store** - The image database that stored images and their associated patient data which also performed backup and retrieval functions.

**X-View** - The software component that allowed the physician to view the x-ray on a personal computer.

**X-Print** - The software component that allowed the physician to print a physical film on demand, if desired.

The developers were assigned to work on one or more of the components. The prototype was developed in procedural code using C. For the new application, the team wanted to use C++ in order to get some of the benefits of programming with object technology. So the team, with what they could learn from the few books on object technology, began to design the software for Project X in an object-oriented manner.

### **2.3 The Schedule**

From the very beginning, the schedule was tight. 10 months is not a lot of time to develop such a large and complex distributed system. The schedule looked something like the following:

Two weeks for analysis

One month for design

Five and a half months for coding

One month for integration and testing

One month for beta trials

One month for bug fixes and final release

A large portion of the total time was dedicated to testing. This was scheduled for the end of the project. This was due to strict quality requirements demanded of medical systems.

The schedule was not a function of a formal project scoping exercise and contained little input from the software developers. Upper management had selected the release date and demanded that the team complete the project by that time. So, the schedule was constructed by starting at the release date and working backwards. No one had the experience to determine if it was realistic or not. But the team was enthusiastic to be working on a cool new product with object technology so they jumped in and began to work.

### **2.4 Development**

After the first two weeks, the Process Analysis Phase was complete although not because the specific tasks were completed that demonstrated a thorough analysis. Analysis was *declared* complete because the time allocated to analysis was over. The team was now chartered to do design.

At the end of the next month, the design phase was complete. Again, not because the team had delivered specific design artifacts; they didn't even have the experience to judge the quality of their own object designs. Design was complete simply because the month scheduled for design was over. At this point in the schedule the team was told to begin coding.

So this team, new to object programming and without any formal training in object analysis, design or C++, began programming. Remarkably, they immediately began to get some benefits from this new object technology. The team set up a system to share objects that they believed were common to the three main non-database applications. They built a class library to support all three applications and rapidly supported changes to the library requested by the other team members. It was, in fact, an exceptional group of developers.

However, all was not rosy. The progress was slower than anticipated and the five and a half months dedicated to coding soon passed. At the end of this period the software was in no way ready for final acceptance testing, much less a candidate for a clinical beta release.

It is important to note that this was the first milestone in the schedule that could actually be measured. The Analysis and Design Phases had no tests to determine if they were complete. They appeared to be complete simply because the date on the calendar had passed. It is only when the project reached the coding stage that it was possible to see how much progress had actually been made. Clearly it was not enough.

Middle management began to worry.

The six months of coding soon became seven, then eight and then nine. It was still not clear to middle management when, if ever, Project X would be completed.

Now upper management began to worry.

Project X had the potential to be an embarrassment to Company Y. Company Y had already announced the new product and had sold it to multiple customers. The customers were expecting it to be installed any day now and the software wasn't even ready for the beta testing process to start.

### ***2.5 The March to Release***

The decision was made to move into the test and clinical beta phase even though new features were still being added. It was hoped that the code would somehow get better by having real users test it. The goal was to get "real mileage" on the application. Product X entered a build, test, code and bug fix phase that lasted for the next three months. The products were installed at test sites and not surprisingly, demonstrated serious quality problems. Major components did not function at anywhere near the quality level required for medical software.

A large database full of bug reports was created. The list of bugs grew at an

alarming rate. Some of the developers had literally hundreds of bugs assigned to them to track, follow-up, and resolve. The developers complained that the testers were logging non-bugs, repeated bugs, and non-important bugs and that a lot of time was being wasted simply wading through the reports. Still, the fundamental truth was that the software was still full of real bugs that would prevent it from being shipped.

### **2.5.1 A Line in the Sand**

At this point, they were now a year into the process and the software was two months late. Many members of the development team had now been working extra long weeks and weekends for almost half a year. The most distressing part was that it felt like the software might *never* be ready to ship. The team, while still in the midst of fixing bugs, was still adding new features.

In an attempt to rectify the problem, the requirements were frozen. Features not yet completed were removed and scheduled for version 2.0. It was decided that fewer features were more desirable than an unshipped system. All of the engineering effort was now dedicated to improving quality and fixing bugs. Time was of the essence, so upper management cancelled all vacations. When several of the team members threatened to quit, upper management backed down. But, the damage to morale had been done.

Thankfully, as the focus shifted to only fixing bugs and not adding new features, the quality of the software slowly began to increase. At this same time upper management announced a "line in the sand" strategy to get the product released. The entire development staff was asked to sign a "line in the sand" document that would guarantee that the new release date, set a few months away, would be met. Morale sunk even lower. The developers were now working 70 hours a week.

### **2.5.2 The Software Release**

This time the target date was met and version 1.0 of the software was installed at several customer sites. Early customers were asked to consider it a beta release and were promised upgrade releases when they became available.

Once the release was installed in the field, the development team focused on continued bug correction. Of course, the field tested the system in new ways and many new issues were reported. Some emergency fixes were required, but most of the issues were simply added to the existing bug tracking database. The systems shipped with 759 known problems.

It wasn't until six months later, with release 1.2, that an acceptable stable release

was finally installed at customer sites. Almost one year to the day later than what was indicated on the original release schedule.

### **3 Project X Successes**

In reviewing Project X, it is important to note that in many ways it was a success. Fundamentally, Project X did finally ship stable software. And it was *working* software that the customers really wanted. Even though it was a year late compared to the original schedule, it was still timely. Company Y quickly became a leader in the field and continues to this day to service this product niche.

There are many practices that contributed to Project X's success. Next we will review a few worthy of special notice. They are:

- Customer participation
- Developing with a small team
- Coordinated object reuse

#### **3.1 Customer Participation**

A strong contributor to Project X's success was the significant customer participation in the development of the software. The basic functionality of the system was specified by users for users. The functional prototype placed key features in front of the users early and the team actively solicited feedback. This ensured that the final system that was delivered provided real value. Customers actually *wanted* the product because it provided useful services.

For Project X, some of the customer participation was also a result of a happy accident. Most of the team members had the opportunity to spend time directly working with customers because the prototype and early releases were so buggy. Developers spent days in clinics and hospitals babysitting the software and locating and fixing bugs. As a result, they got to know the users of the system personally: from the patients, to the technicians to the diagnosing physicians. The users were not an “abstraction” to the team any longer; they were Joe, Mary, Mark and Dr. Probert. The developers often called and talked directly with the users and the users called and talked directly with the developers. As a result, the team learned valuable lessons from the users first hand. They learned that the users were not sophisticated computer engineers interested in learning about complex user interfaces. They were not impressed with lots of buttons, cryptic icons and clever features. Rather they were busy medical professionals trying to diagnose real medical problems for real hurting patients.

It is essential that software be developed with real users in mind and not in an engineering only vacuum. The Standish Group studies determined that having

users directly involved in the software development process is the single most significant factor in a project's success.

Development teams must have personal contact with the end users. They must understand how the software will improve the user's life. Visit the customers. Learn how they do their work. Understand from personal experience how your software will make their life better, and then build that system.

Project X software delivered value to X-Ray Departments because the development team knew the departments' needs intimately. This was a significant contributor to the software's long-term success.

### **3.2 Developing with a Small Team**

A second contributor to Project X's success was that it was staffed with a small and committed team. Because the team was small, communication between team members was natural and non-threatening. Each team member felt directly responsible for the success or failure of the project and the entire team worked hard to deliver it.

The size of the team is very important in object software development. It is not acceptable to simply take 50 or 100 Cobol programmers and place them on an object project. Large team sizes that may have historically been employed for conventional programming projects and processes are not appropriate for object projects. In fact, any object development team that contains more than ten members is immediately at risk.

It is often preferable to work with only six to eight developers on a team. If the project is larger and really requires 100 developers then find a way to break it into ten smaller projects with 10 people.

Large teams are not conducive to development on object projects. This is in part because at the core of the object development process is the building of object models. It is essential that the entire team understand the model that is being developed. If the team is too large, it is simply impossible for all of its members to have an adequate understanding of that model. Due to the power and flexibility provided by object development it is expected that a small dedicated team of 10 to 15 engineers can easily outperform 60 engineers working in procedural development environments.

### **3.3 Object Reuse**

A third significant contributor to the success of Project X was the concept of Object Reuse. Object Reuse is the long promised, seldom-delivered benefit of

object-oriented development. The idea is simple. Other team members can reuse a class once it has been written. In practice, however, it is unusual to see large numbers of classes shared across multiple applications. This was not the case though for Project X.

From the very beginning of development, Project X engineers identified something they called "common sources." "Common sources" represented the classes that the development team believed could be shared across the three major applications. By the end of development, the team had over 200 classes being shared as "common sources." This was an incredible amount of reuse for a team that had never before developed object-oriented software.

How was this accomplished? Well, when a team member wrote a common class it was almost immediately submitted to a shared repository. Because the entire team was on an aggressive schedule, the new common classes were typically checked out and used by other team members within a week. It was common of course, that someone took issue with how an object was originally coded. Feedback was immediate. This aggressive reuse schedule resulted in the common classes being rapidly evaluated, criticized and reworked to accommodate the design ideas of the other developers.

As a result, the common classes were of a higher quality than would ever be expected from a novice team. The process also made the Project X team members seasoned designers in a very short time.

#### **4 Project X Failures**

Of course, as was clear from our earlier synopsis, Project X was not a string of pleasant triumphs upon triumphs. There were many major mistakes in project management that made the project unpleasant, longer and more costly than necessary. In fact, at the end of the project, Company Y privately considered the project an overall failure.

In this section we will review three major management mistakes that occurred on Project X:

- Using a waterfall process with a long lead time for delivery
- Not managing requirements effectively
- No formal training or education program

##### **4.1 Using a Waterfall Process**

The Project X development process was described as follows:

- Two weeks for analysis
- One month for design
- Five and a half months for coding
- One month for integration and testing
- One month for beta trials
- One month for beta fixes and final release

This style of laying out a development process is known as “Waterfall.” Graphically laid out, it looks much like an actual waterfall. At the top of the waterfall is the Analysis Phase. Waterfall, as it is often practiced, assumes that we can do a complete job of analysis for the entire project in the Analysis Phase. At the end of the Analysis Phase, it is assumed that analysis is complete and we can move into the Design Phase. Occasionally, out of the Analysis Phase will come some sort of analysis report. The publication of this artifact is the signal that the Analysis Phase is complete.

The Design Phase works in a similar manner with a discrete beginning and end. The Design Phase is followed by the Implementation Phase and the Implementation Phase is followed by the Testing Phase. At some point after the Testing Phase a software application pops out of the process.

Project X experienced several problems by following this process. The first is that it is often difficult, if not impossible, to know that the artifacts produced in analysis and design are accurate or complete. In Project X these phases were considered complete simply because the time allocated for them had passed. The engineers, if asked, would admit that they really did not consider them adequately completed. However, given that the engineers were working with new tools, processes and techniques they probably never would consider analysis complete.

The fact that they were not done adequately however could not be proved until the implementation was complete. It is only at this time that the software can actually be measured and tested. It is usually only at the end of implementation, late in the entire process, that management first becomes aware of problems. Of course, at that point, it is often too late to redirect the effort effectively.

It is important to note that the Waterfall process used by Project X appeared to be a reasonable process to implement. It emphasizes upstream planning, analysis and design. Surely it is a good idea to design software before sitting down and writing it. The Waterfall process identifies necessary activities and orders these activities in a logical manner. It makes sense to do analysis before design, design before implementation and implementation before testing. After all, this type of Waterfall process is how bridges, dams and other building are built. Surely this should be a good way to build software, shouldn't it?

In fact, a Waterfall process is often used to successfully deliver small software projects. A small project is one that can be delivered in two weeks to two months by a small team of two to five people. If this describes your project then this type of Waterfall process may work for you.

However, this process is likely to be disastrous for larger projects delivered over longer periods of time. Consider a project large if it involves over 10 engineers and long if it last over six months. Project X would classify as a long project because of its 10 month implementation schedule.

#### **4.1.1 Waterfall Weaknesses**

There are several reasons why using the Waterfall process in this manner doesn't scale for large projects. Practitioners make the wrong assumptions. Here are just two:

1. It assumes that requirements do not change. It assumes that once the Planning Phase is complete, requirements are locked down. In fact, requirements change all of the time.

Requirements can change in longer projects as the hardware and software available change. Requirements can change as the competitive markets change. Requirements can change as users interact with early versions and offer new suggestions. Requirements can change as we learn more about our users and their needs. Therefore, a single requirements phase is almost never adequate for long project.

2. It assumes that we can get design right the first time. Although almost all Waterfall processes tolerate going back and reworking design as limitations are discovered. In practice though, it is often considered the exception, not the rule. This process expects developers to get design right the first time. This is a problem because developers are almost always being asked to work with new languages, tools, techniques and processes.

With all of these new concerns it is almost impossible to get a good design the first time through. Team members simply don't have all of the information. Also for the developers on Project X, it was the first time they had ever developed an application using object technology. There is no way they were going to get a good design on the first pass and the process contained no mechanism to allow design to be revisited.

#### **4.1.2 Fixing the Process**

In object oriented programming, the Waterfall process can be improved by making it incremental and iterative. An incremental process is a process that

seeks to repeatedly implement small yet complete capsules of functionality. The functionality is formally released at the end of an increment.

In the ten month schedule to deliver Project X, they should have broken down the project into 10 to 25 increments. At the end of each increment, the team would be responsible for delivering a completely functional working piece of the system.

Incremental development would have helped Project X in at least two ways:

1. The early increments would have most likely failed to deliver the desired features. Thus, it would have made it clear to everyone early in the Implementation Phase that the team could not hit such an aggressive delivery schedule.
2. It would have made it significantly easier to manage the feature creep that threatened the final release. This is because the features that are scheduled are explicitly evaluated at the beginning of each increment. There is no way to sneak features into the system without the agreement of the entire team including the developers, and marketing and sales representatives.

At the end of any given increment a question can then be asked, "Do I have enough functionality to release the system to real customers?" If the answer is "yes" then you can ship the software to users. If the answer is "no" then ask "What is the next most important piece of functionality to add in order to be able to release the product?" Then that functionality should be scheduled in the next increment.

This type of planning would have saved the developers from spending days and weeks on features that were not included in the final release. Iterative development says that at the beginning of each increment, it is acceptable and expected to do more analysis and design. In fact, often in incremental development, the old code is "refactored" or improved simply to make it more readable and understandable. This can be done because the process of building the application teaches the developers ways to better design the system in the next increment. Think of it this way, Project X was a large project to deliver in ten months. It really needed 10 months of analysis, 10 months of design and 10 months of implementation. The only way to possibly fit that into 10 elapsed months is to do them simultaneously, in an incremental and iterative development process. The actual functionality that will be included in the first release to be shipped will become clear as the project progresses.

An iterative process is a process that recognizes that the act of developing software will uncover new insights and information that will cause additional

analysis and design. There are many graphical models of an iterative process, and most of these have flaws. But all of them recognize the fact that in each increment, the entire Waterfall process may be revisited with additional planning, analysis and design done specifically to support that increment. There are techniques that exist to help manage the incremental and iterative development process. The most important of which is using Use Cases or User Stories to drive the entire process. We will examine this next as we review the second failure of Project X: poor requirements management.

#### **4.2 Poor Requirements Management**

The project manager captured the software requirements in a document called a PDR (Product Definition and Requirements). However, this document was not complete at the beginning of the project and additional features were added to it even up to the point when management declared a "line in the sand."

During the course of implementation, the developers added features and code based on what *they* decided that *they* wanted to implement next. The addition of code and features was in no way coordinated until late in the development cycle and even then the only coordination was of tracking bug reports and bug fixes. Project X needed a better process for planning code development and change requests.

##### **4.2.1 Use Case / User Story Driven Process**

Project X would have benefited greatly from a Use Case driven process.

Use cases are stories that describe how the system is to be used by real people to produce some valued result. Use Cases will define how users interact with the system. For every complete course of events initiated by a user we will identify one Use Case. Use Cases are best created by analysts observing real users in their own environments.

Project X should have started out by identifying all of the users who need to interact with the x-ray system and all of the Use Cases that could be performed by those users. Then during planning, the Use Cases should have been ordered by importance. Thus, the most important Use Cases could be identified and implemented first.

If Marketing is careful in which order the Use Cases are to be implemented, they may find that they may have a working system that can be shipped months before they thought possible. At the end of each increment, new Use Cases may be added, old Use Cases may be deleted and the order of which Use Cases should be done next may be changed. Note that the developers never set the

order. The customer of the application, who in this case is the Marketing Department, always sets the order.

Use Cases are created at the very beginning of the development process during the initial business modeling process and should be used as the basis for all additional activities to follow. They are not to be abandoned when the requirements are completed because they will direct the analysis and design. They are not abandoned when analysis and design are completed because they will define what will be implemented. Use Cases are even useful after implementation is completed because they provide the framework for the functional test plans. Even after the test plans are completed, the Use Cases should be saved since they serve as a wonderful template for training programs and user manuals.

If the project manager for Project X had employed Use Cases in an incremental and iterative process, there is little doubt that the product would have shipped earlier, at less cost and with higher quality.

### ***4.3 No Formal Training***

Another mistake Project X management made was not to provide any formal training for the development team. The team had no commercial experience delivering object-oriented systems. They would have benefited significantly from training in object analysis, object design, object project management and object languages. The team would also have benefited from specialized training in their target operating system. None of this training was provided.

Professional technical training is costly and time consuming. It is common for managers to say that developers cannot spend two weeks in training classes because this is not enough time. This is simply nonsense. There is an old saying "If you don't have time to do it right; how will you have time to do it over?" Training is about learning how to do it right.

If you don't have time for the proper training then you don't have time for the project. It is not uncommon for the time spent on training a team in object technology to be returned seven fold. At the time Project X was developed, there were not many training opportunities with regard to object technology. That is no longer the case. There are no excuses for an ill equipped team.

### ***4.4 Line in the Sand***

It would be incomplete to review the project management failures of Project X without talking about the "line in the sand" strategy. If you recall, late in the project while the team was in crisis mode the team members were asked to sign a

document that drew a figurative line in the sand. This said effectively that the software must ship by a certain date.

As a motivation to the developers, the "line in the sand" strategy was a failure. It was seen as an insult and as yet another demonstration of clueless management. The developers were not consulted to select the date and hitting the date was not a commitment the developers understood. It was clear to them that the "line in the sand" strategy was actually no strategy at all but rather just another arbitrary release date. It was as if management believed that the developers really weren't trying that hard to make the date. By insisting that they sign such a document, management was simply trying to shame them into meeting the deadline.

However, destroying developer morale aside, the "line in the sand" ultimately did help to ship the software. It stopped Marketing from adding new features and forced the team to focus simply on making the existing features work correctly. Not adding new features was probably the single most important decision to help Project X ship. Ultimately, shipping the software was more of a function of the number of bugs that would be tolerated rather than the amount of features the software included. Thus, the project met the "line in the sand" release date, in spite of it being arbitrary.

### **5 Non-Happy Endings**

As stated at the beginning, Project X was a muddy mixture of failure and success. At the end of the project no one in Company Y was happy with what had been delivered or the process that was used to get there.

Project X succeeded because of customer involvement, the small-dedicated team and the practice of extensive object reuse. Project X failed because they didn't implement a process that could be adequately monitored or controlled. Project X failed because it implemented the wrong process, without requirements control and without proper training for the team. Of course, there were lots of other smaller problems in Project X; such as having the test team located 2,000 miles away from the development team.

In the long run, the software was shipped and the customers experienced a significant improvement in the way their X-Ray Departments operated. Exams were done faster, at a higher quality and at a lower cost.

Six years later, when one of the customers was visited, it was surprising to see how happy they still were with the system. This was ironic since Company Y seemed to think that the whole project was an embarrassing failure.

In a final expression of that feeling of failure a new manager hired in by

Company Y fired almost the entire original development team. Of course, almost all of those who weren't fired quit. And so Company Y ended up with a product in the field and almost no-one left in the company who knew either the product, the market; or the lessons learned. A year later the manager who made that clever move was also released from Company Y. Later still, Company Y hired back many of the original developers.

Software development can be trying and painful, but doesn't need to be. Don't repeat the mistakes made by the team members of Project X. Implement an iterative and incremental development process, employ Use Cases or stories to drive that process and train your staff.

If you do, your project *will* succeed.