Project Management for the 21st Century

Mark D. Lincoln

Introduction

As the software industry has grappled with the problem of delivering quality software on time and within budget, it has attempted to use many techniques to manage the software development process. The growth of object-oriented programming and rapid application development is where the software development industry has concentrated their efforts. Although these techniques have yielded improvements in the speed at which software is developed, they have not had the expected beneficial effect on the quality of the resulting software. Why is this?

Is Software Development Art or Engineering?

I hear this question debated at every technical conference I attend and inevitably, someone always chimes in with, "it is a little of both". The real question is "what should software development be?" By posing the question in this manner, we can postulate possible methods for making software development less of an art and more of a formal engineering process.

The opponents to formalizing software engineering see this as an effort to stifle the creativity in the software development industry. It is, however, an effort to stifle the chaos currently growing in the software development organizations around the world. Formalizing software engineering will mean that we will all follow a set of standard processes on the way to delivering the highest quality product possible. This effort will raise software development to a level on par with the techniques used by engineers in similar disciplines like building construction, without any loss of innovation.

Is Software Development Easier?

Many of my colleagues point to rapid application development environments like CA-Visual Objects, Visual Basic, Delphi, Jasmine Studio, and Opal as a testament to the ease with which we can now develop software. What they fail to take into consideration is the underlying complexity that these environments hide from the developer using the tool. Although the development of simple applications or prototypes has become easier, the environment in which the application executes grows more complex with every passing day.

For instance, if a developer would like to create a simple application comprising a single screen to perform a simple task, the rapid application development environment makes this very easy. It is this ease with which it handles simple tasks that causes the problem. If the problem that the developer needs to solve strays from the ordinary and simple, then the developer must dig deeper into the tool, or more importantly, into the operating system to solve the problem.

Many assume that one can simply bolt together objects or components in order to solve any presented problem. While this may seem like the future of software development, we have not arrived at this point yet. Many third-party companies, however, would want you to believe that this is the case. They call the new breed of software developers that assemble high level components into finished software applications, "composers" or "assemblers". The problem with this scenario is the fact that these high level developers lack the basic software development knowledge to deal with the unusual low level problem. This means that some other software

developer must handle the low-level issues that arise in virtually every software development project. Therefore, since the environments for which software is being developed are becoming more complex, the process of developing software for these environments also remains highly complex.

Are We Building a Better Mousetrap?

Anytime I am asked if we build better software today versus even ten years ago, my answer has to be an emphatic, "No". This may puzzle many of my colleagues, but you only need to ask the average home or business user of a personal computer whether they consider the software they use today more reliable than that of ten years ago, and you will have your answer. Some users even continue to use the same word processor that they used ten years ago because it has all of the features they need and does not crash as often.

If it is the case that the software that we develop and that others use is inferior to the software that was developed many years ago, why do we not see a wholesale revolt by the end users of our software? Well, the easiest answer is that the end users have been beaten down by years of having to put up with, "Good Enough Software". They suffer from the "boiling frog" syndrome where the software they use gets slightly less reliable with each new release. If all of the problems appeared in one release, they would simply not use it, however, this incremental degradation causes them to get used to the old problems by finding different ways to do their work. Is this the way things should be in our industry? Obviously not, but we have yet to provide a means to create the quality in our software that will raise the level of satisfaction from our users.

To put this in historical context, you only need to think about the hardware and software technology that the National Aeronautics and Space Administration (NASA) used to land a man on the moon. Compared to the tools we have access to today, they accomplished this Herculean effort using "stone knives and bearskins". The modern personal computer has over a thousand times the computational power of the computer in the lunar module. They were also working toward a goal that would not excuse mistakes. Failure to deliver could mean the failure of the mission and possible the death of the crew. This doesn't mean that they never made mistakes, however, the quality of their work at a time of limited technology is impressive to say the least.

Are We Better Programmers?

If we could land men on the moon with such antiquated technology, why can't we deliver high quality software on time and within budget? Were the programmers of that era better than the programmers of today? Were they more dedicated and more innovative? Simply put, the problem of software quality lies on several fronts. I do not believe that the modern programmer has a problem with dedication or innovation. Actually, I think we have quite the contrary situation. We are seeing unrivalled technological innovation especially in the software arena.

What the modern programmer does suffer from is a lack of software development discipline. This is a trait that has been fostered and nurtured by the ability of the individual programmer using a personal computer to create software. It manifests itself in poorly designed software with little or no documentation both at the source level and of the design and implementation.

During the mainframe software heyday, it was very difficult for a programmer to become a self-taught software developer. The era of the personal computer, however, changed all of that.

Many of my colleagues purchased a personal computer for doing their finances, word processing, or email, and then became software developers by reading books and through trial and error. Although one can learn to program this way, one will lack the type of insight provided by a more academic background. Software solutions come from anecdotal information and through trial and error, but rarely are such solutions optimal in design or implementation.

Although this type of software development is typically not optimal, it does have a positive side. Software developers not constrained by the "right" way to do things often offer very imaginative and innovative solutions to problems. This means that the project managers have a far greater responsibility today than in years past. They must nurture the imagination and innovation of the modern programmer without sacrificing the quality of the software that they are developing. Unfortunately, since many projects are faced with pressing deadlines and floating requirements, this does not happen, which leads to the chaos.

Organizations demonstrating this characteristic of chaos in software development in some ways resemble the cattle towns of the old west. The may have a management structure that attempts to monitor the development of their software, but for the most part, these managers function as the "town marshals" attempting to keep the cowboys from destroying the town on their night off. They let the cowboy programmers do as they please as long as they stay within some sort of arbitrary yet ineffective structure. The "cowboy" programmers develop software using their own techniques and favorite tools without regard to the ability of the organization to maintain or enhance their creation.

What exacerbates this situation even more is the level of turnover for technical personnel in our industry. There is no guarantee that any one developer will see the development of a piece of software through from design to implementation. This makes the need for better management techniques even more important. The project manager may be the only member of a software development team that survives the entire project.

My Patch, Your Patch, Which Patch?

One of the techniques used for fixing software that has been delivered in a substandard condition is to issue patches (a.k.a. service packs, services releases, updates) to the software. Several companies currently support this niche technology with offerings that enable a software development organization to issue patches periodically as their end users report flaws, defects, bugs, shortcomings, or "undocumented features".

Although this technology works relatively well, it causes many headaches for the end user who must apply one or more patches to make his software work as it was advertised to. If the end user fails to apply the patches in the correct order, omits one, or worst of all, the patch itself has an undocumented side effect, the end user can be left with an environment that may barely function at all. (At the time of this writing, I just applied the latest update to my Web browser and my laptop now fails to boot up properly every other time I turn on the machine. Many attempts to correct this side effect of applying the update have, to this point, failed.)

This scenario would not happen to any user of our software if we would deliver a better quality product at the initial release. I am not saying we should be expected to deliver perfect or bug free software, just that we should do better than the, "Good Enough Software", that requires several service packs before it is considered stable and reliable.

Is There a Solution?

The solution to the chaos in a software development organization is a set of formalized processes that every development project follows regardless of the size, scope, or complexity of the project. Implementing formal software development processes in an organization can be painful because the "cowboy" programmers will resist standardization. In the end, however, most of them will realize the benefits of higher quality software with less brute force effort, which means they can spend more time thinking of new innovations for future projects.

Requirements Management

Since the requirements provided by the client or end user of the application are the driving force for any software development project, a software organization must implement a process for requirements management. Failure to do this puts any organization at a disadvantage when attempting to deliver a solution to the end user. Without requirements to compare with the finished product, it is difficult or almost impossible to discern if the software meets the needs that the end user outlined at the beginning of the project.

Very often, a client or end user will not have any written requirements from which a valid design can be created. This is a very real problem facing many organizations attempting to provide custom software for clients. It is imperative that the software development organization creates the requirements from meetings with the client or end user. Then the documented requirements must be presented to the client for their approval before the development organization can design the software. This enables the client and the software developers to both understand what the finished product should do.

Several third party vendors provide tools for requirements management. A software development organization may also choose to develop a tool that fits their specific needs. In any event, the tool should contain the ability to manage, track, and update requirements for individual features or modules in a project. This will enable the project manager to track the ability of the new software being developed to meet the needs of the end user. Actions can be taken during the development of the software to handle any deficiencies found so that they do not slip into the final product requiring a service release or patch.

Design Documentation

After completing the requirements step in a software development project, the developers need to create a design that meets the requirements and then document the design. It is amazing how many projects skip this step and go directly from requirements to implementation. This is where modern rapid application development environments cause problems.

Because it is extremely easy to have a conversation with a client or end user and then start to produce the user interface for an application, many software projects jump right into development. Skipping the design, however, means that the amount of time spent reworking the modules of an application takes away from the time needed to produce a quality piece of software.

Therefore, it is imperative that the team creating the software spends the time to create and document the design before development begins. This design should also be presented to the client or end user so that they are comfortable with it. If they have comments or suggestions at

this point, it is okay to alter the design to meet their requests. After the client or end user has approved the design, then development can begin.

Now, if you have a very large project with many autonomous modules, it is possible to perform the design incrementally. The development team can design each module, get approval from the client or end user, and then begin the development of the module. This incremental design approach works best when the software utilizes an architecture with several autonomous components. If there is a tight coupling or dependency between the components or modules, it may not work, which will require completing the entire design before implementation.

Project Estimation

After the requirements and design documentation is complete, the time and resources necessary to complete the project must be estimated. This is one of the tasks that all software developers and project managers have a very difficult time performing with any degree of accuracy. As a result, many clients and end users do not respect the estimates that we give then when planning a project. They typically expect our estimates to be low and that we will not meet our project delivery date. This is why many clients require a fixed price for development projects since they realize that paying hourly or daily for the project that exceeds the time estimate will cost them far more money.

Therefore, it is imperative that you come up with a system for estimating your projects that not only uses your experience, but also makes use of the results of previous projects. This way even if your estimates based on experience and judgment and low, the empirical data you gather from your completed projects will force you to become more accurate in your estimation.

We have come up with an estimation system that involves the entire development team. Each member including the project manager produces an estimate for the project. This involves breaking the project down into its constituent modules, components, or features and producing an estimate for each part of the project. The entire development team, as a group, should perform the breakdown of the parts of the project. Then each individual performs the estimation process. At the conclusion of the estimation, the team reassembles and finalizes the overall estimate for the project.

Each member of the team uses an estimation technique that involves a three step refinement process. During the first estimation step, you estimate the time, in hours that it will take to implement each individual part of the system using amounts that are multiples of ten hours with nothing taking fewer than ten hours no matter the complexity of the feature, module, or component. The estimated times for the parts of the project are typically ten, twenty, forty, or eighty hours, but the estimate can be any multiple of ten hours. If you find that you have estimates for individual parts of the project that are extremely large (greater that 120 hours) then you should attempt to decompose them into smaller parts. This will help the accuracy of your final project estimate.

After the first round of estimates, each individual on the project development team then refines their rough estimates for each part of the project twice more using judgement, experience, and (if available), empirical data gathered from previous projects. This results in three sets of estimates with each succeeding set of estimates more refined than the previous set. Once each individual comes up with their project estimates, the entire team meets again to discuss, debate, and finalize the estimates for each module, component, or feature of the project. The finalized estimates are

then entered into the project plan and tracked so that they can be used for future project estimation efforts.

Status Reports

Each of the members of the project development team should create a periodic status report for the project manager on their progress and any outstanding issues they may be facing. The period that the status reports cover is arbitrary, but it should not be too long. I prefer weekly status reports. It keeps the material in the reports fresh in the minds of the developers and also allows them to spend very short periods of time preparing the report information.

The format of the reports is also arbitrary and you should decide on a format that works best for your organization. I prefer reports that have a bulleted list of tasks that the developer is working on or has completed. This simplifies the report and insures that the developers are not spending a lot of time preparing text for the report. After all, we want the developers creating software, not essays.

Each developer must assign a status code to each item in their report. The status codes should identify each item as "not started", "in progress", or "complete". I have found that assigning percent complete values to status items as meaningless and arbitrary. For example, it is very difficult to differentiate between something being 80% complete and 90% complete, so I have discarded this type of metric in favor of the previously mentioned status codes.

The administrative staff on the project should enter the information in the status reports into a project tracking system. They should also place the status report documents into the source code control system so that their information can be extracted and analyzed later. This type of information becomes invaluable when going through the process of estimating time and resources for similar projects in the future.

Status Meetings

Depending on the time interval you choose for your status reports, you should choose a similar time interval for having meetings to discuss the reports. Each developer should present the highlights from their report. These meetings have great benefit, especially if a developer is having a problem either with the design or the implementation of a component or module. It enables a developer to discuss possible solutions with the other developers as well as the project manager. The team has a greater ability to solve the problems together and this keeps everyone informed on the items or the components that the developers can share.

Project Planning and Tracking

All projects regardless of the time involved in the implementation should have a project plan that outlines the time and resources allocated to the project. The project manager and the technical staff assigned to the project should prepare the project plan in your project tracking tool of choice. I prefer Microsoft Project as it offers all of the features we need as well as good integration with the Microsoft Office products. The administrative staff should maintain the project plan based on input from the project manager and the technical staff assigned to the project. This input should come in the form of time reporting, the previously mentioned periodic status reports, and the status meetings.

The administrative staff should include the project plan as part of the items under version control for the project. This will enable the project manager to use the completed project plan as an estimation tool for future projects with similar requirements. It should contain invaluable information about the time and resource estimates originally made for the project as well as the time and resources actually used during the implementation. Without this type of quantifiable information, project estimation becomes an exercise in anecdotes and guesses. With the numbers from the project plan, the organization can eventually reach of point of giving very accurate time and resource estimates.

Project Debriefing Reports

After the team completes the project, each member of the team including the administrative staff and the project manager should prepare a short report on the entirety of the project. The format of these reports is also arbitrary, however, I find it is best to let everyone write a description of their impressions on their part of the project as well as on the entirety of the project.

These reports provide valuable insights into how the team worked together as well as how the team accomplished its goals. They should also become part of the project information under version control along with the other reports and documents. They become valuable when assembling teams for future projects. They also help the project manager to avoid some of the problems encountered on the project, enabling the software development organization to become better with experience.

Project Debriefing Meetings

After the members of the development team have prepared their respective debriefing reports, the project manager should have a final meeting to discuss the contents of the reports. The enables the entire team to discuss their impressions of the project as well as sharing ideas for future projects. The project manager should be careful to control the meeting especially after a difficult project. Sometimes, the team members will have a tendency to vent their frustrations and while this can be a good thing in moderation, the team members will have to work together again, so moderation is the key to avoiding bad feelings.

Coding and Documentation Standards

In order to promote uniformity in the source code and source code documentation, your organization should create a set of coding and documentation standards for all current and future projects. This will be one of the areas that your developers will resist. Coding style tends to be a personal thing, however, if your organization is to build the highest quality software, this must be done. In the end, your developers will thank you for this, especially when they have to work on source code that was created by another developer.

Since the technical staff in an organization may have a significant amount of turnover, a common coding and documenting style will insure the success of the project. Every developer will be able to work with every other developer's source code since it will be familiar to them. This also insures that modifications and enhancements are simpler since any experienced developer in your organization can make the changes.

Code Reviews

In order to promote an atmosphere for developing quality software, the project manager and technical staff should perform periodic source code reviews with the intent of helping the implementation team for the project to improve the final product delivered to the client. These code reviews should not be used as a means to criticize the developers. Their purpose will be to insure that the development staff on the project is using the latest techniques and standards so that the final product delivered to the client exceeds the expectations of the client.

These code reviews can also be made part of the periodic status meeting where one of the developers on the team presents a new technique, class, or component. This will let all of the technical staff on the development team see the item in use and described by the person who developed it. This can also be a time where a developer presents the code under development to solve a difficult issue. This enables everyone on the team to offer possible solutions to the issue.

Source Code and Configuration Management

Since most development projects involve multiple developers, it is imperative that your organization uses some type of system for source code control. This enables many developers to work on the implementation simultaneously without concurrency problems. These systems also enable your organization to keep all of the components of a project together for efficient retrieval and use. These components can be things like the requirements document, the design document, the status reports, the project plan, the debriefing reports and all of the third party components used in the project.

Source code control also becomes very important when you decide to enhance the software for a future release. It makes it very simple to recover your development environment as a starting point for the new enhancements. You also have the benefit of having all of the information from the previous development effort for planning the new development effort.

Testing and Quality Assurance

This is the step that amazingly gets left out on most projects that have tight timelines in order to make the "all important" release date. The thought here is usually, "if we get something, anything, out by the deadline, we have a success." Unfortunately, this often results in the client doing the testing of the delivered software. This is not good since the client will obviously be dissatisfied with the low quality and instability of the software.

I subscribe to the philosophy that any software we create is, "better late than lousy". This means that we go through a complete testing and quality assurance cycle to insure that we deliver the highest quality product possible. In order to do this efficiently, you should have a formal test plan in place. This may or may not include automated testing software. The key point is that your quality assurance staff should have knowledge of good testing techniques including a clear understanding of regression testing. Having the developers test the software is not sufficient since the developers will test the software to see if it works while a good tester will test to see if it breaks.

Staff Development

On of the biggest problems facing our industry is the fact that the technology changes so rapidly that it is extremely difficult to stay up with all of the current and emerging technologies. This is where the development of the staff in your organization is important. I give the staff the

opportunity to be involved in their own development. First, I encourage each of the staff members to spend ten to twenty percent of their time doing something that advances their technical knowledge. This time amount is always dependent on their current schedule and whether they are billable or not. Most of them use this time to research new technologies, read books or technical journals, or develop components that they may use in future projects. Second, I like to give each staff member the opportunity to attend either a technical conference or a formal training course. The knowledge gained during either of these endeavors can be invaluable to your organization.

The great fear of many organizations in that if you give your staff too many opportunities to enhance their skills, they will leave your organization for another opportunity. If you let this fear dictate what you allow you staff to do, then you will not get any benefit from enhancing the knowledge of your developers. My best advice here is to ignore this fear. You will have to deal with staff turnover regardless of this.

Conclusion

Since our industry is not building top quality software, we need to make some changes to reach a point where we are delivering the software that our users need and everyone wants. Although the transition from where we are to where we want to be is not simple or easy, there are some steps that any organization can take to arrive at their goal of delivering higher quality software on time and within budget. I have given some of the areas that my organization has concentrated on to make this transition and we have had good success. If the software industry is to move from software development as an art to more of a formal engineering process, we have to start somewhere. If your organization implements the steps I have suggested, you should see a noticeable improvement in the quality of software you develop.

Biography

Mark D. Lincoln is the President of Cycle Consulting, Inc., a professional services firm specializing in database development using Delphi, CA-Visual Objects, Java, and C#, Internet E-Commerce applications, and GPS communication systems. He has been a featured speaker at various international technology conferences and writes for several technical publications. Mark has been programming in CA-Clipper since 1987 and is an avid teacher of object-oriented techniques using CA-Clipper, CA-Visual Objects, Delphi, Jasmine, Java, and C#. He can be reached via email at mdlincoln@compuserve.com.