ABSTRACT

The discipline of project management has been available for almost 40 years. This paper examines several factors that make the management of software development projects more difficult than the management of many other kinds of projects. These factors negatively impact the software project managers’ success in estimation and scheduling, planning, responding to schedule and budget pressures, monitoring project status and managing the integration, and testing phases of the project. Techniques for addressing each of these difficulties are described, and topics that need to be incorporated into software project management courses to address these issues are outlined.

INTRODUCTION

Project management techniques were initially developed to assist the managers of large, complex military and aircraft system development projects in coordinating the resources and tasks required for what are typically multi-organization and multi-year projects. Since the initial development of these techniques in the late 1950s and early 1960s they have come to be used in numerous industries. The techniques are now used routinely in various parts of the construction industry, in the pharmaceutical industry, in the software development industry, as described by Thayer (1997) and Royce (1998), and
in many others. Great success has occasionally been reported as a result of the use of project management methodologies, particularly in the construction industry. A Guide to the Project Management Body of Knowledge (PMBOK) has been developed under the auspices of the Project Management Institute’s (PMI) Standards Committee (1996). This body of knowledge has served as the basis for numerous courses and workshops offered by academic, industrial, and professional organizations. And the PMI sponsors a Project Management Professional (PMP) certification program.

Despite the facts that many software developers and managers have participated in project management workshops and courses, and that many software projects employ at least one person who is designated the “project manager” (some of whom are certified PMPs), software development projects have continued throughout the last 40 years to be over budget, take longer than expected and sometimes not provide the level of quality and functionality expected by users. Numerous publications such as Brooks (1975, 1987), Van Genuchten (1991) and Jones (1995) support this assertion.

The application of standard project management techniques to software projects has not achieved the success noted in some other industries because typical software development projects are fundamentally different from projects in other industries. These differences make the management of software projects more difficult than the management of some other types of projects. Many of these fundamental differences have to do with what we will call “visibility” throughout the remainder of this paper. By visibility we mean the ability of the project manager to confirm completion of a task simply by looking at the results of that task. The lack of visibility that the software project manager has into various tasks of a typical software development project handicaps that manager in his or her attempts to do a good project management job. Only by being aware of these differences and by developing an ability to address them can the manager have any hope for success.

The following sections of this paper address an area in which I believe there is a significant difference between a typical software project and other types of projects. Several examples will be drawn from the construction industry to illustrate these points. Methods for addressing each of these factors will be described and topics that must be successfully taught to software project management students to implement these methods will be outlined. The methods that have been successfully used by the author to teach these topics are described briefly. Figure 1 outlines a view of a typical large software development project. It represents a simplified combination of the traditional
waterfall model as well as a more modern incremental approach to software development. In Figure 2 we have overlaid indicators of where in that model the subjects covered in each section of this paper would be principally applicable.

Many of the ideas expressed here and the methods used for teaching these topics were developed while I was teaching a project management workshop for software developers in AT&T and Lucent Technologies during the period from 1994 through 1997. During that time inputs from and discussions with over 1600 workshop participants led to the conclusions and opinions described in this paper. After the recommended topics were incorporated into that workshop and used by several of the workshop participants on industrial projects, the author validated the applicability of these ideas in numerous project reviews in which he has participated since 1995. Unfortunately, because the reviewed projects were not done under well-controlled experimental conditions it has been possible to get only subjective

Fig. 1. Combined waterfall and incremental project model.
feedback on the value of these approaches. It has not been possible to develop quantitative evaluations of the value of each of the recommended methods.

**ESTIMATION AND SCHEDULING**

Anyone who has been through the building of a home or a major home remodeling in which the services of an architect has been used will recognize the following description. Very early in the process an architect interviews the homeowner to determine a list of needs (requirements). Then the architect asks a very important question about the homeowner’s budget for the project. The architect leaves and develops some preliminary plans that meet as many of the needs as possible while remaining within the customer’s budget. The architect will typically use an estimate of cost per square foot or per cubic foot to produce an early estimate of the project’s cost. They may use a different per unit cost for new construction versus renovated areas and different costs for
basement areas, main living areas, kitchen areas or bathroom areas. This early estimate is similar to early software project estimates based on counting numbers of features or function points as described by Albrecht (1979), Jones (1986) and Matson, Barrett, and Mellichamp (1994). Then, after one or more rounds of feedback from the homeowner the architect refines the plans and updates the cost estimates. When a plan and a cost have been accepted by the homeowner the architect will proceed to develop a set of construction prints from which a building contractor can develop cost and schedule estimates and actually construct the building.

At that point the homeowner, or the architect, will usually seek price and schedule proposals from one or more building contractors. It is not unusual for the homeowner to be shocked when the contractor returns with cost estimates that are considerably higher than the original budget. At this point another round of discussions among all three parties may take place during which a revised plan is developed in order to bring the cost closer to the original budget. Only when the homeowner is satisfied with the new plans and the corresponding cost estimates, provided by the contractor will a contract be signed, and construction begin. While construction is underway the homeowner usually decides to make additional changes which may change the cost.

In many ways the development of a new software system or product is similar to the construction of a house. Requirements are developed. Early (rough) estimates of schedule and cost are made and requirements, cost, and schedule are modified until the customer is satisfied with the high level requirements, the cost estimate and the approximate schedule. Then the detailed architecture, the project plan and design work begins.

One problem with this scenario as compared with the house building scenario is that, while a residential architect has tools and notations at hand to visually describe the new home in terms (floor plans and elevations) that laymen can understand reasonably well, the software requirements developer and the software architect are relatively limited in the notations that they can use to present a high level proposal to the customer. Most depend on written English to describe these concepts to the customer. A few can afford the luxury of developing a prototype to illustrate some of the proposed system capabilities. Some visualization techniques, like UML (Unified Modeling Language), are beginning to make inroads into this area, but such techniques are far from the notations that are available to the building architect and are still not well understood by potential software system users.
In addition, there is no well-accepted cost per feature or cost per function point measure that software development managers can use as the basis for early cost estimating. However, some tools like COCOMO, which was originally defined in Boehm (1981) and reformulated in Gulezian (1991) and other parametric models are beginning to be used for this purpose.

What does all of this mean for our students? It means that our software project management students need to be aware of the problems and pitfalls of this rather fuzzy front-end process. They need to learn how to efficiently develop prototypes for user interfaces or use visual tools such as UML in a way that is both precise and in a way that their customers can understand. They need to learn how to move from high level requirements to a detailed architecture and to a corresponding project plan that serves as the basis for a firm and final cost estimate. And, most importantly, they need to learn how to conduct negotiations with the customer that are similar to those conducted among the building architect, the homeowner, and the contractor.

I have used three methods to teach these topics to students in a reasonable amount of time. First, a brief lecture on what function points are and a description of a set of rules for counting function points has been presented. This is followed by a simple example in which the instructor works with groups of students to actually count the function points, from a simple set of requirements, for a small software application that might be developed by a single individual. The instructor then explains what COCOMO and some of the other available parametric models are and shows the results of an estimate developed using a parametric model. Second, the instructor explains what a bottoms-up, or task-based, estimate is and, using a case study for which the architecture and the tasks to be accomplished are provided to the student, the student has an opportunity to develop a detailed estimate for a somewhat larger software development project. And, finally, a role-play case study is used that requires the student to negotiate the transition from an initial (low) estimate to a more sound (higher) estimate which, according to the case that was designed for this purpose, takes place over a period of several months, but which in practice is compressed into a few class sessions.

**PLANNING—WHAT, HOW, WHO, WHEN AND HOW MUCH?**

Proper project planning, as defined in references such as Constantine (1993) and Thayer (1997), requires that a project plan document be produced that answers the questions:
• What is the team planning to do?
• How is it going to be done?
• Who is going to do each task?
• When will each piece be done?
• How much will it cost?

If a plan does not contain answers to these questions, it is not a plan. The most valuable portion of the plan is the process that key members of the team go through to produce such a document. That process provides a great opportunity for team members to get to know each other and to reach agreement on the plan. Unfortunately, for many software development projects a plan that answers all of these questions is never produced. Some teams feel that because they have been given requirements by the customer, because they have a standard development process, such as those described in Humphrey (1990) and in Chroust (1996), in place that describes how the work will be done, because it is too early to determine who will do each task and because writing all these down in a presentable form will take so long, the “real work” of the project will never get started. The usual result, when these are the prevailing opinions of core team members, is that only a high-level work breakdown structure and a high level precedence diagram gets produced. This is what the team refers to as their project plan.

To produce a credible project plan that answers all of the questions outlined above takes intense effort. I have found that the best way to produce a draft of a project plan for a real software development project is to remove planning team members from their work environment for a few days to produce a preliminary plan. During that period members of the group will develop a list of high level requirements. They will outline, in as much detail as possible, the tasks that need to be accomplished in the form of a work breakdown structure. They will determine the appropriate relationships among the tasks, make a first cut at who, or at least what kinds of people, will do each task, estimate the duration of each task, schedule those tasks and produce an overall project schedule. While some members of the team are working on the details of linking and scheduling tasks, other members of the team are writing drafts of the textual and budget sections of the plan. For modest sized projects (10–50 team members) most of this work can be accomplished in a week. For larger projects more than one dedicated period may be required for the development of a complete preliminary project plan.

After a preliminary plan is produced the team can return to its normal work environment and the remaining details are worked-out and documented by
team members during the next few weeks. The resulting plan does not usually look like a literary masterpiece. A good technical document is usually sufficient. It does not need to be a large (several hundred pages) document. Small projects usually require small plans (10–20 total pages). Larger projects require larger plans.

Most students in a software project management course do not believe they can produce a credible project plan in such a short period of time with intense effort. One of the pieces of a course that I have found to help in making them believers is to give them an exercise of developing a preliminary project plan for a project that will require 10–15 staff members over a period of 18–24 months. It will typically take a group of 7–8 students who have some reasonable level of software development experience only 2–3 days of full-time work to produce a plan for a project of this size, even when none of the team members has significant experience with a similar project. Therefore, I believe that all software project management courses need to contain an exercise that allows the students to experience such a planning process, primarily to convince them that it can be done efficiently with this level of effort and in a relatively short time. In practice, I have used a case study during a one-semester graduate level course that occupies students for approximately 1 1/4 hr of class time per week and which results in a fairly detailed project plan in the 13th week of a typical 15 week semester. This gives the students the experience of developing such a plan and very effectively convinces them that the task can be done in just a few days of dedicated full-time work in an industrial work environment.

SCHEDULE AND BUDGET PRESSURES

Software development teams are often put under great pressure to underestimate the cost and duration of their projects. This happens first when the project manager is asked to give a preliminary estimate of the time and cost that will be required—sometimes even before the functional system requirements and the architecture are specified, and almost always before a detailed project plan has been developed. In my experience most software developers are naturally optimistic. Frequently when they are asked for early estimates they will underestimate the size of the job.

Later, when they know the requirements, have spent enough time to develop a solid architecture and have put together a detailed project plan they
frequently need to revise their estimates upward. When they do that their own management, their sales team and frequently the internal or external customer will say something like, “You have already given me estimates that are half that size. You had better figure out how to do this job for something close to your earlier estimates, or we will have to find someone who can do the job better!”

The project manager is now faced with a dilemma. Should he or she “give-in” and say, “We will try to find a way.” Or, should they stay with their now, hopefully sound, estimates and risk not getting the job. All too frequently the manager “folds” and postpones facing the sound and fury of the customer and his or her management. Inevitably, the day of reckoning comes at some future date when the team needs to own-up to how long the job will really take and how much it will cost to deliver all of the originally specified capabilities with an appropriate level of quality.

This is a real ethical issue for software developers. I have found that the best way to address this issue is to always strive for impeccable ethics. When early estimates are requested (or sometimes demanded) the team leader needs to develop an estimate based on one of the commonly available parametric models, such as COCOMO, and present these estimates along with the assumptions that were used for developing the estimates, such as the assumed function point count, system complexity, development team capabilities, etc. During the interval when functional and system requirements, architecture and a project plan are being developed the manager needs to revisit the parametric estimates frequently and discuss them with management and/or the customer. When a detailed project plan is finally available the resulting estimates, once again, need to be revisited and discussed with the stakeholders. This slow process of educating the stakeholders about the costs and time required for what they are requesting is the only way I have seen project managers successfully navigate through a project from beginning to end.

In order to instill in our students both the necessity for and capability to carry out such a process we need to teach them something about business ethics. We also need to teach them about parametric estimation modeling and the pros and cons of the various models, and we need to give them some knowledge of and practice of developing good task and network-based project plans. Finally, because the time required for each task in the network is not known precisely, there is usually significant uncertainty about the time when the project can be completed. Students need to learn how to analyze and
manage task-based networks in which the time associated with the tasks is probabilistic.

I have made use of the same role-play case described in the Estimation and Scheduling section of this paper to stimulate a follow-up discussion of the ethics that might be involved. During the time between the role-play and the next class meeting I have typically asked students to prepare written answers to some of the following questions based on their experience in the budget and scheduling negotiations.

- Should you give in to the customer’s demands for a shorter schedule and lower costs and worry later about how to achieve that commitment?
- Should you “pad” the schedule to take into account the uncertainties associated with task durations?
- How should you handle estimation of time and costs for tasks that you have forgotten to include in the project’s work breakdown structure?

Based on the student’s prepared responses to these questions I have conducted a facilitated discussion, with most of the input provided by the students, that typically leads to the importance of integrity and continuous communications with the customer throughout the project definition phase of the job. This discussion provides an excellent opportunity to introduce various codes of ethics into a classroom setting where a straightforward discussion of these ethical codes would be somewhat abstract and probably boring.

**PROJECT MANAGEMENT AUDITS**

After all of the work outlined in the previous section is completed the organization of which the project team is a part, the manager of that organization and the project manager’s boss need to have some way to verify the soundness of the plan. The organizational manager could conceivably sit with the project team to review its assumptions, read the details of the project plan and verify that the team has taken everything into account that should be taken into account, and has properly calculated the resources that are required and their costs. However, most real-life organization managers simply do not have the time, the patience or the cognitive capability to do this task single-handedly.

To assist the organizational manager with this verification activity some software organizations have begun to conduct routine project management
audits shortly after a detailed project plan has been developed. Project management audits were first described by Bernstein (1981). Since that time very little has appeared in the literature about project management audits. Bernstein’s initial description defined a project management audit as a tool to be used by the organization to determine why a project had gotten into trouble and what needed to be done to fix the problem. However, the problem with that approach was that once a project had gotten into trouble that was serious enough to be recognized outside the project team it was already too late to do anything to get the project back to its original objectives. All that an audit could do at that point was to minimize the damage caused by a project that was already a failure. Or, sometimes, to kill the project because so much damage had already been done.

In order to prevent projects from getting into serious difficulty project management audits should be conducted shortly after a detailed project plan has been developed. These audits are conducted by a small team of experienced managers who have had experience in managing software projects similar to those being done by the team being audited. They do not need to be technical experts on the problems being addressed by the team. But, at least one of the audit team members should have had experience as a member of an audit team. Prior to the audit the project team provides the auditors with the project plan, with any requirements that have been developed and a description of the proposed system architecture.

The audit team, which is usually made up of four or five experienced managers, visits with the project team for three or four days. They start by hearing a series of informal presentations on the project, its objectives, its technical structure, and its project plan. They interview a wide cross section of team members and provide a verbal report, usually documented in a series of slides that outline the positive things that they found, a list of concerns that they have about the project and, for each concern, a set of recommendations about how they would address that concern if they were managing the job. This feedback is given to the project manager, perhaps to the entire project team and, sometimes, to the project manager’s boss. The information is for those people to use as they see fit.

The purpose of the audit is, first, to keep pressure on the team and the project manager to complete the plan by a well specified date. The second purpose is to find improvements that could be made that would result in lower cost, a shorter schedule or an improvement in the probability that the project will be successfully completed. Finally, the audit provides assurance to the
organizational management that the project will be successfully completed either with or without the suggested improvements.

In order for students to be receptive to the use of post-planning project management audits they need to be provided with some evidence that the use of audits tends to improve the likelihood of success. They need to be taught how to receive and use audit recommendations. Finally, they need to be instructed in how to conduct an audit, through both presentation and practice. In particular, they need to learn how to listen to a technical presentation, ask appropriate questions that are appropriately worded, how to conduct one-on-one interviews with project team members and how to synthesize their findings and recommendations.

In a classroom setting I have used the project plan developed during the first 13 weeks of a class as a basis for a demonstration of how a software project management audit should be conducted. If there is more than one plan developed in the class I would typically select a small number of people from each of the teams to be auditors for one of the other teams. Alternatively, if the instructor is able to recruit experienced software development managers to audit the plans prepared by the students the experience of being audited during the last 2 weeks of a course can illustrate to the students the operations that are necessary to conduct an audit and the value that the planning team can get out of a well conducted audit.

MONITORING PROJECT STATUS

It is extremely difficult for a software project manager to determine and define the precise status of a project, particularly during the phases where requirements, architecture, design and code are being developed. The reason for this difficulty is that the project manager cannot practically determine when each task associated with these phases of the job has been completed. Because for large projects it is difficult, if not impossible for a software project manager to personally review and evaluate each artifact that is produced during these phases the manager usually has to depend upon verbal reports from the person doing the work to determine whether the work has been completed. Software developers tend to be optimistic when reporting status of their work. As a result the manager frequently hears that the task is 90% complete.

The manager needs some very objective way to determine whether or not a task has been completed. In the construction industry it is fairly easy for the
homeowner or the general contractor to visit the building site and visually confirm that a particular task has been completed. For more complicated portions of residential construction for which the homeowner does not know quite enough to determine if a task has really been completed they can rely upon the successful completion of building, plumbing and electrical inspections conducted by the municipal government where the construction is taking place to determine with high probability that preceding tasks have been completed.

Software inspections were first proposed and used by Fagin (1976) at IBM. The purpose of software inspections is, nominally, to find faults in software artifacts. That objective has been well described by Dyer (1992), Gilb and Graham (1993) and by Strauss and Ebenau (1994). Several studies, such as Ackerman, Buchwald and Lewski (1989) and Porter, Siy, Toman and Vota (1997) have shown that the use of carefully conducted inspections is economically justified for both improving quality of the final software product and reducing the cost of development of the software.

Inspection results can also help a project manager monitor the status of individual software development tasks. For each artifact that the development team decides to inspect the project manager can use the results of the inspection much like the homeowner can use municipal building inspections to determine when a particular task required for the construction of his home has been completed. When the output from the software development task has been successfully inspected by a team of peers the project manager can usually assume with relatively high probability that the task has been successfully completed. The visible indicator is the inspection report indicating that the inspection team has signed off on the work. If requirements, architecture, design and code documents are regularly inspected successful completion of the inspection of these documents indicates that the task has been completed. This provides the project manager with visible evidence that the task has really been completed.

To assist our students in managing these activities requires that they learn what inspections are, how to manage an inspection program, how to do inspections of those items for which the project manager is responsible and how to use inspection results to monitor projects during those phases where there is typically less visibility of task completion.

For students who have had experience in conducting inspections, either through a course or from some simulated project experience, it is quite easy to make the points outlined above in a lecture and that requires the students to
specify as part of their semester-long project planning exercise how and where inspections will be used for project monitoring. However, if students have not previously learned these techniques, it will be important to teach through some experiential activity what they are and how to do inspections. I have found that a combination of a short lecture on inspections combined with a video that shows an inspection taking place followed by some classroom practice doing an inspection of a short requirements document is enough to provide students with the background they need on inspections. This needs to be followed by a requirement that they build the process into their semester-long project plan.

INTEGRATION AND TESTING

Very frequently, when students are learning to write small to medium sized programs they fall into a pattern that is currently being strongly recommended by some members of the computer science and software engineering communities. That is incremental development and testing. A student develops a small version of the ultimate application, then incrementally adds features, doing some minimal testing along the way. Then when all capabilities have been developed some overall feature and system testing is done.

Ideally, large, multi-person development teams should follow a similar procedure. However, larger software development teams frequently do little or no planning of the integration portions of the project. They seem to assume that 15 people can independently develop their own increments and when all are finished the piece parts are added (sometimes simultaneously) to whatever base exists, all at one time. The usual result of this kind of integration is chaos. This is frequently the first time that the project manager realizes that the project is in trouble. Efforts to integrate the disparate pieces drag on well beyond the time when system and overall functional testing is scheduled to begin.

What is needed during this interval is, first, a plan for systematically combining and testing the work of individual team members, two at a time or, at most, three at a time. Then the team needs to follow this plan (or a modified logical version of it to account for the possibility that some deliveries may be late and others may be earlier than initially scheduled) until all the piece parts have been integrated. At each step during this interval at least some limited testing must be conducted to determine which piece parts need to be corrected.
or modified to function properly. In order for the manager to determine the status during this sometimes-protracted integration interval frequent demonstrations, based on the integration tests should be scheduled as part of the plan. When an individual reports that an integration increment has been successfully completed that individual should volunteer, or be requested, to demonstrate one or more of the tests that led the individual to conclude that the integration step has been completed. Demonstrations starting at this point provide the project manager with the same kind of visibility that is available to the homeowner who can look at a completed foundation, framing or wall finishing in a new home. It is typically the first point on a large project when the project manager can see for himself or herself the visual evidence that a task has been completed successfully.

Likewise, during functional and system testing, frequent demonstrations of selected test results provide visual evidence that test work is progressing on, ahead or behind schedule. These demonstrations, of course, do not prove that all faults have been eliminated. But they do provide enough information for the manager to confirm that the work is progressing well or where additional assistance might be needed to resolve some knotty problems.

In order for our students to learn how to manage these phases of the project they need to first learn about testing principles. They need some practice in developing integration and test plans and finally they need to learn how specific integration, system and functional tests should be selected for demonstration. This is more material than an instructor can usually include in a project management course. If the curriculum in which the student is participating contains a testing and quality course that is the best place for the student to learn most of this material. However, I have found that a short lecture on testing principles in a project management course that emphasizes how testing should be planned and quantitatively tracked is a good refresher. It also helps prepare the student for developing those portions of the class room project plan that cover testing. When I have given such a lecture I emphasize the fact that one of the most common project management audit concerns that I have observed the complete lack of an integration plan. I then provide a one-figure illustration of what is meant by an integration plan and tell the students that the auditors of our class project plans will be paying special attention to the integration part of their plan. This warning, combined with a few questions from the instructor when doing intermediate reviews of progress on the plan, usually results in a reasonable integration plan being included in the classroom plan.
When we teach software project management to computer science students we need to cover many of the subjects contained in the PMBOK. However, in addition to those standard topics, the instructor also needs to include appropriate materials on both parametric and task-based software estimation. The importance of the development or use of an appropriate software development process as the foundation for the “How” portion of the project plan must be emphasized. Business ethics and negotiation skills associated with requirements, budgeting and scheduling activities need to be included. Appropriate use of requirements, design and code inspections, not only as a quality technique, but also as a method for monitoring project progress requires that students become knowledgeable about how to do and how to manage inspections. Finally, the importance of integration test planning and the use of frequent demonstrations during both the integration and system testing phases of the project need to be emphasized.

To appropriately add all of these topics to the most important topics included in the PMBOK certainly represents a challenge for the instructor. However, these topics are at least as important as many of the basic topics included in the PMBOK. In fact, they may be more important than some of the PMBOK’s basic topics. The software project management instructor’s real challenge is to determine what basic topics can be omitted in order to include those outlined in this paper.

While I have had some very good experiences in teaching the topics outlined in the previous sections of this paper I have never been successful in fitting all these materials into either a one-semester course or into a week-long intensive workshop for experienced software developers. In my opinion it would probably require a two-semester course to take someone with no prior project management education or experience through the most important PMBOK topics in addition to the topics described in this paper. I have had some success in delivering most of the material outlined here in a one-week intensive workshop to experienced software project managers. In developing and teaching both industrial and academic software project management courses I have omitted some PMBOK topics what some instructors of traditional project management courses feel are very important. These have included topics such as earned value, critical path calculation algorithms and statistical quality assurance. I believe that it would be helpful if there were a forum in which software project management instructors could have extended
discussion of which traditional project management topics could be omitted in order to add material on the topics outlined in this paper.

Perhaps this paper will serve to stimulate further discussion of this topic.

REFERENCES


