UNITED NATIONS

E

1974 ~ 1999
ESCWA

ECONOMIC AND SOCIAL COUNCIL

Distr.
LIMITED
E/ESCWA/TECH/1999/WG.2/9
6 September 1999
ORIGINAL: ENGLISH

Economic and Social Commission for Western Asia

Expert Group Meeting on Project Planning
and Management in Research and
Development and Quality Assurance
Beirut, 21-23 September 1999

TRENDS IN SOFTWARE DEVELOPMENT
A PROJECT MANAGEMENT PERSPECTIVE

Trends in Software Development - A Project Management Perspective

Frank R. Parth

Unlike managing pure research projects with their high degree of technological uncertainty and risk, managing software development does not have the same amount of risk associated with it. If we can come up with the idea, we can create the software to implement it, the only things we need discuss are how long it will take and how many people are required. Yet the history of project management in the Information Technology (IT) arena has a poor success rate. Most software development projects (numbers as high as 85% have been quoted) fail to deliver the system to the original requirements and within the original time and budget predictions.

While the early success rate is poor, the situation is improving even though projects are getting larger and more complex. This is because the practitioners of project management are learning better techniques faster than the field of software design is changing, and so are able to manage software projects better than we could even five years ago.

Why Is SW Management So Difficult?

While the basics of SW project management are no different than the basics of other project management, there are some fundamental changes in emphasis that reflect the different nature of software development.

Unlike hardware development such as construction projects, it is not obvious until very late in the project whether or not the code meets the requirements. A building that does not look like the plans can be immediately identified. A software developer may or may not interpret the requirements in the way that the user intended. Unfortunately, a software module that does not do what the requirements state is often not identified until very late in the development cycle.

Also unlike construction project management where everyone can see progress being made, software development takes place in peoples' heads, and it is virtually impossible to measure progress when you can't see what's going on.

A third difference is that unlike hardware development, testing the product and integrating it are neither simple nor obvious. Once a SW developer has looked at the requirements and interpreted them in his or her mind, he or she will design and code to that interpretation and will test to make certain the software satisfies that interpretation rather than whether it satisfies the requirements.

Why is software development changing so fast?

In addition to these fundamental differences in SW project management from other project management, in the past few years there have been many changes in how software is developed that need to be accommodated by appropriate changes in how we manage that development. We can no longer manage software development the same way we just a few years ago.

The primary reasons for this are both internal to technology itself and external to the environment. Technology-driven reasons include:

- The development environment has changed rapidly and will continue to change rapidly, new languages (such as Java), new operating systems (such as Linux), and increasingly fast hardware have added a great deal of complexity to the development environment,
- There are many new tools available to make the programmers more efficient in writing code (from Rapid-Application-Development tools to 4th generation languages),

they find it very difficult to put themselves in a position of someone who does not have nearly their capabilities and level of comfort with software. Programmers make assumptions about what the users are capable of that are generally far too high.

In addition to being "too experienced" to properly interpret the requirements, programmers become programmers because they want to solve complex problems by creating good processing code. They are generally not interested in building screens and buttons for the users. But it is the screens and buttons that the users will use, so that is the part they have the most interest in. We find ourselves in a common situation where the most important part of the program, the user interface, is the least interesting to the programmers. It is often put off until the very end of the coding and done quickly in order to meet schedules.

The solution to these two problems is to have someone represent the users (in all of their ignorance) and design the user interface first. If you have a team of programmers, it is often helpful to split the team into two groups - one team would write the user interface and the other team would write the code that actually does the work. Only when the person representing the users is comfortable that the application is easy to use and not confusing should the job of designing the background, processing part of the program begin.

This processing code, in addition to the user interface, is critical to the success of the program. Not only must it function correctly, the code must be able to run on multiple versions of an operating system as well as not interfere with other programs that the users might be running and on the newest hardware as well as older hardware. This is a time-consuming and difficult task. Since few programmers have the wide variety of experience needed to understand all of the technical requirements, it has been found helpful to hold regularly-scheduled code review meetings where other programmers with different knowledge can review a design and offer insights and suggestions.

Building the project plan

We said earlier the second improvement that can be made in managing software development is to build the project plan based on the experience of the programmers in whatever development tools and languages will be used.

Good project management techniques have always dictated that the project schedule should be driven by the amount of work that needs to be done to meet the requirements, not by a deadline date picked without regard for whether it can be met or not. This still holds true in software project management. The people who will be doing the work are the best determiners of how much effort it will take to meet the requirements.

Software developers, however, tend to be optimistic if asked to estimate the overall project schedule. While their input to the schedule is absolutely necessary, they tend to underestimate the amount of time required to determine and document the requirements and to test and to integrate (if necessary) the final product. The project manager needs to work very closely with the developers to identify the specific tasks needed to build the project plan and the relationships among those tasks.

Modern software is generally coded in individual modules which are then linked together to form the overall application. When asked how long a module will take to code, developers tend to be optimistic. They will estimate a task's duration based on a standard work day rather than on how much time they are actually productive during a day. Experience has shown that developers are actually working on their task about 75% of the time. The remainder of the time is spent in meetings and other non-task-productive work. If a developer estimates a task to take four days to design, code and module test, that task should be written into the product schedule as five days to account for the non-productive time.

Another item the project manager needs to ask the programmers is their level of familiarity with the development tools being used. Many programmers will tell you they can quickly learn any new tool and be productive quickly. However, every tool needs to be learned and the programmers will need to spend time learning it if they are not already familiar with it. There are many new development tools on the market and programmers always want the latest tools. The project manager needs to build additional time into the schedule early if a new tool is being brought in, even if the later tasks can be shortened as the developers become familiar with the tool and more productive with it.

Independent Testing

Historically, software developers were also the people doing the testing of the final product. This can be a problem. Nobody knows the design and the code better than the person who wrote it, and they are the best people to test the code at the unit/module level to make certain it works. However, developers will always test their code to verify that it works as they designed it. They do not test the code to make certain it meets the user's requirements or to ensure that
the users will not crash the program under normal user operating conditions.

This is a significant difference in approach which can be resolved by giving the testing to a group independent of the developers. The purpose of the testing group is to test the final code to make certain it meets what the users asked for (so they need to be very familiar with the requirements as well as with the design of the code) and to make certain that in a simulated user environment the
code does not cause problems and will run with the expected operating systems and the other applications an average user might have on their computers.

Properly testing is much longer process than most developers think and most project managers would like. However, the benefits of testing far outweigh the cost and schedule impacts. It is far cheaper to find problems during testing than it is to release a product and have the users find significant problems. A commercial software package will lose some market share if it is released late. If it is released too soon and the users find significant problems, the product will never capture any market share and can ruin the company's reputation in the process.

Summary

The software development field is changing rapidly because of advances in technology and environment pressures. There has never been a greater emphasis in meeting tight schedules while holding down costs. Project managers become the focal point for everything and receive the greatest pressure of anyone on the project. If the project is successful, the whole project team receives the credit. If the project fails, the project manager often receives the blame alone. By understanding the differences between software development projects and other types of project management we can significantly increase our ability to deliver the project on time, within budget, to the requirements needed.

- Increasing size and complexity of software make the management of requirements much more important and increase the need for a thorough testing program,

Environmentally-driven reasons include:

- Applications have grown increasingly complex, with users demanding far more functionality than they did a few years ago,
- There is a major change in emphasis from the pure processing part of the software to the user interface,
- Time-to-market is critical, the faster a company can put products into the market the greater their ability to capture market share,
- Development costs are very high, so cost management is increasingly important,
- There is an increased emphasis on developing software, which is easily upgradable and flexible,
- There is also an increased emphasis on aligning internal product development with the business goals.

Improvements in Project Management

In managing software development projects early we simply used the approaches and techniques that work in other projects. For example, early attempts to manage SW projects predicted the schedule based on some estimate of how many lines of code a developer could write or, later, how many function points could be completed in a day or a week.

While this approach is attractive, it has proven to be worse than useless because it gives the appearance of managing when, in fact, we are not. The differences in size between inefficient code and well-written, terse code can be as large as ten to one. Poor code can be written faster than good code, and the differences are not seen until the very end of the project when the code fails testing. Unfortunately, at this point the impacts of poor code are devastating to the project cost and schedule, because not only does the code itself have to be rewritten, but all code interfacing with it also has to be modified and the full range of testing repeated.

The solutions to the problems of software development lie not with changing what the programmers are doing, but in changing how the project is set up and managed. Programmers have more efficient tools to design programs and write code. It is no longer sufficient for a software project manager to sit at a desk and watch 5 programmers working at their computers and assume that everything is fulfilling the project plan.

There are three major areas where changes need to be made. The first is to put new emphasis on defining the requirements that the software must satisfy. The second is to build the project plan based on the experience of the programmers in whatever development tools and languages will be used. And the third is to set up an independent testing program so that the testing and integration work is done by people other than the programmers themselves.

Defining the requirements

With the increasing complexity of application programs one of the most crucial areas that needs to be managed is the definition of the users' requirements. It would not be unusual when planning the project to allocate one-third of the project time to defining and understanding the requirements, one-third to designing and coding the software, and one-third to performing the testing required. The more time that can be spent up front understanding exactly what the users' needs are, the less time is wasted at the end of the project redesigning, recoding, and retesting software to meet what the user thought they asked for in the first place.

Of all the personnel involved in a software product, the programmers have proven to be the least accurate in understanding what the users want. It is not because they don't want to understand the requirements, on the contrary, they want very much to understand and to deliver a product the users will be happy with. The problem occurs because programmers have so much more experience with computers than their users do that