ISSN 2229-5518

Applied Software Project Management Software Project Planning Estimation Techniques

T. Rajani Devi

Abstract - Most critical activities in the modern software development process is without a realistic and objective software project plan, the software development process cannot be managed in an effective way. The purpose of project planning is to identify the scope of the project, estimate the work involved, and create a project schedule. Project planning begins with requirements that define the software to be developed. The project plan is then developed to describe the tasks that will lead to completion. software and project estimation techniques existing in industry and literature, it has strengths and weaknesses. Usage, popularity and applicability of such techniques are elaborated. In order to improve estimation accuracy, such knowledge is essential. Many estimation techniques, models, methodologies exists and applicable in different categories of projects. None of them gives 100% accuracy but proper use of them makes estimation process smoother and easier. Organizations should automate estimation procedures, customize available tools and calibrate estimation approaches as per their requirements.

Key Words- Black art, business domain, fair estimate, granularity, magnitude, magnitude estimate, quibble, rough estimate, starved, weighing factors.

1 Introduction

SOFTWARE project management begins with a set of activities that are collectively called project *planning*.

Before the project can begin, the manager and the software team must estimate the work to be done, the resources that will be required, and the time that will elapse from start to finish. Whenever estimates are made, we look into the future and accept some degree of uncertainty as a matter of course. Software project planning actually encompasses several activities planning involves estimation—the attempt to determine how much money, how much effort, how many resources, and how much time it will take to build a specific software-based system or product. The appropriate software is for everyone in the project to understand and agree on both why and how that software will be built before the work begins. That's the purpose of project planning process cannot be managed in an effective way. Project planning is an aspect of Project Management that focuses a lot on Project Integration. The project plan reflects the current status of all project activities and is used to monitor and control the project. The Project Planning tasks ensure that various elements of the Project are coordinated and therefore guide the project execution.

Project Planning helps in - Facilitating communication - Monitoring/measuring the project progress, and - Provides overall documentation of assumptions/planning decisions. The Project Planning Phases can be broadly classified as follows: -Development of the Project Plan - Execution of the Project Plan - Change Planning is an ongoing effort throughout the Project Lifecycle.

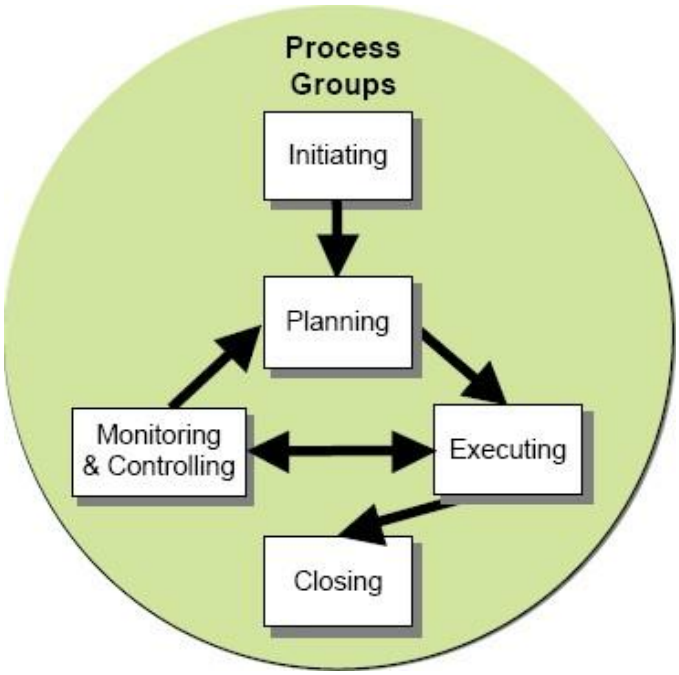

Fig 1: project life cycle

2 Objectives

The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost, and schedule.

These estimates are made within a limited time frame at the beginning of a software project and should be updated regularly as the project progresses.

In addition, estimates should attempt to define best case and worst case scenarios so that project outcomes can be bounded.
The planning objective is achieved through a process of information discovery that leads to reasonable estimates.

3 Useful Estimation Techniques for Software Projects

3.1 The Importance of Good Estimation

Software projects are typically controlled by four major variables; time, requirements, resources (people, infrastructure/materials, and money), and risks. Unexpected changes in any of these variables will have an impact on a project. Hence, making good estimates of time and resources required for a project is crucial. Underestimating project needs can cause major problems because there may not be enough time, money, infrastructure/materials, or people to complete the project. Overestimating needs can be very expensive for the organization because a decision may be made to defer the project because it is too expensive or the project is approved but other projects are "starved" because there is less to go around.

In my experience, making estimates of time and resources required for a project is usually a challenge for most project teams and project managers. It could be because they do not have experience doing estimates, they are unfamiliar with the technology being used or the business domain, requirements are unclear, there are dependencies on work being done by others, and so on. These can result in a situation akin to analysis paralysis as the team delays providing any estimates while they try to get a good handle on the requirements, dependencies, and issues. Alternatively, we will produce estimates that are usually highly optimistic as we have ignored items that need to be dealt with. How does one handle situations such as these?

3.2 We provide reliable estimates

Programmers often consider estimating to be a black art—one of the most difficult things they must do. Many programmers find that they consistently estimate too low. To counter this problem, they pad their estimates (multiplying by three is a common approach) but sometimes even these rough guesses are too low.

Are good estimates possible? Of course! You just need to focus on your strengths.

3.3 What Works (and Doesn't) in Estimating

Part of the reason estimating is so difficult is that programmer can rarely predict how they will spend their time. A task that requires eight hours of uninterrupted concentration can take two or three days if the programmer must deal with constant interruptions. It can take even longer if the programmer works on another task at the same time.

Part of the secret to good estimates is to predict the effort, not the calendar time that a project will take. Make your estimates in terms of *ideal engineering days* (often called *story points*): the number of days a task would take if you focused entirely on it and experienced no interruptions.

Ideal time alone won't lead to accurate estimates. I've asked some of the teams I've worked with to measure exactly how long each task takes them. One team gave me 18 months of data, and even though we estimated in ideal time, the estimates were never accurate.

Still, they were *consistent*. For example, one team always estimated their stories at about 60% of the time they actually needed. This may not sound very promising. How useful can inaccurate estimates are, especially if they don't correlate to calendar time? Velocity holds the key.

3.4 How to Make Consistent Estimates

There's a secret to estimating. *Experts automatically make consistent estimates*[1]. All you have to do use a consistent estimating technique. When you estimate, pick a single, optimistic value. How long will the story take if you experience no interruptions, can pair with anyone else on the team, and everything goes well? There's no need to pad your estimates or provide a probabilistic range with this approach. Velocity automatically applies the appropriate amount of padding for short-term estimates and risk management adds padding for long-term estimates.

There are two corollaries to this secret. First, if you're an expert but you don't trust your ability to make estimates, relax. You automatically make good estimates. Just imagine the work you're going to do and pick the first number that comes into your head. It won't be right, but it *will* be consistent with your other estimates. That's sufficient.

Second, if you're not an expert, the way to make good estimates is to become an expert. This isn't as hard as it sounds. An expert is just a beginner with lots of experience. To become an expert, make a lot of estimates with relatively short timescales and pay attention to the results. In other words, follow the XP practices.

4 Useful Estimation Techniques

Before we begin, we need to understand what types of estimates we can provide. Estimates can be roughly divided into three types:

- **Ballpark or order of magnitude:**
  Here the estimate is probably an order of magnitude from the final figure. Ideally, it would fall within two or three times the actual value.
- **Rough estimates:**
  Here the estimate is closer to the actual value. Ideally it will be about 50% to 100% off the actual value.
- **Fair estimates:**
  This is a very good estimate. Ideally it will be about 25% to 50% off the actual value.

Deciding which of these three different estimates you can provide is crucial. Fair estimates are possible when you are very familiar with what needs to be done and you have done it many times before. This sort of estimate is possible when doing maintenance type work where the fixes are known, or one is adding well-understood functionality that has been done before. Rough estimates are possible when working with well-understood needs and one is familiar with domain and technology issues. In all other cases, the best we can hope for before we begin is order of magnitude estimates. Some may quibble than order of magnitude estimates are close to no estimate at all! However, they are very valuable because they give the organization and project team some idea of what the project is going to need in terms of time, resources, and money. It is better to know that something is going to take between two and six months to do rather than have no idea how much time it will take. In many cases, we may be able to give more detailed estimates for some items rather than others. For example, we may be able to provide a rough estimate of the infrastructure we need but only an order of magnitude estimate of the people and time needed.

4.1 Doing an order of magnitude estimate

This is what most of us face when starting off a new project. New technology, teams unfamiliar with the technology or domain, or unclear requirements ensure that this will probably be the best estimate we can provide.

- Break the project down into the different tasks needed. Try to get as many tasks as possible. A useful way to break down tasks is to consider typical software activities such as analysis, design, build, demo, test, fix, document, deploy, and support and see if they are required for each task and whether they need to be broken out into new tasks.
- Evaluate each task on two scales: complexity (high, medium, low) and size of work (large, medium, and small). A less complex task may still involve a large amount of work; for example, loading a database with information from paper forms may take several weeks. A very complex task may not involve much actual work but can still take a lot of time, as in tuning a database for optimum performance. Complex tasks are usually hard to split between many people/teams while large-size, less complex tasks can usually be split up between many people/teams.
- Tasks effectively fall into one of nine combinations of complexity and size. For each combination, define an expected amount of time and resources required. For example, we could say that low complexity and small-size tasks will take one week at most, medium complexity and small-size tasks will take three weeks, and so on. These weighing factors will differ based on the team and project and should be reviewed after the project to help get better values the next time. Add together all these values for each task to get an estimate of time and resources required.

Size	Complexity		
	Low	Medium	High
Small			1) Tune database
Medium			
Large	1) Load data		1) Integrate with security system 2) Create data validation routines

Fig 2: sample table for doing order of magnitude estimate

4.2 Doing rough and fair estimates

These estimates can be done when you have a good idea of the tasks to be done and how to do them.

- Those who will do the actual work are the best people to do these estimates. One then can add up all the estimates from different people to get the final estimates.
- Ensure you collect estimates on the three variables of time, people, and infrastructure/material needs.

- Break down tasks to as detailed a level as possible. As mentioned previously, it can help to consider typical software activities such as analysis, design, build, demo, test, fix, document, deploy, and support and see if they are required for each task. Break tasks down to a granularity of eighty hours or less.

5 Conclusion

Here the preceding techniques can help one achieve better estimates. Review estimated needs versus actual needs after every project. identify what was correct and what was wrong. This will help you improve the next time. As with many other activities, experience will help you get better!

REFERENCES

[1] Software Engineering-A Practitioner's approach, Roger S.Pressman (5th edi), 2001, MGH

[2] Software Project Management, Walker Royce, 1998, Addison Wesley.

[3] McConnell. Steve," software project Survival Guide"(very thin but to point)

[4] Lauesen, Soren," Software Requirements: styles and Techniques", good comparative overview of requirements techniques

[5] Futrell, Shaffer," Quality Software Project Management" (extremely thorough and well-written)

[6] Pierre Bourque and Robert Dupuis, ed (2004). Guide to the Software Engineering Body of Knowledge - 2004 Version. IEEE Computer –1. ISBN 0-7695-2330-7. http://www.s

[7] Lewis R. Ireland (2006) Project Management. McGraw-Hill Professional, 2006. ISBN 0-07-147160-X. p.110.

[8] Young-Hoon Kwak (2005). "A brief history of Project Management". In: The story of managing projects. Elias G. Carayannis et al. (9 eds), Greenwood Publishing Group, 2005. ISBN 1-56720-506-2

Authors Details

Hi, I am T. Rajani Devi, I have done M.Tech (S.E) from Kakatiya Institute of Technology & Science, Warangal. I have worked as Assistant Professor. I have six years of experience at academic side.