Estimating Software Development Time

I recently had to sit in a talk of how to estimate the duration of a software development task. They came up with the usual trivial stuff everyone would come up when thinking about it as well as some more complicated even including formulas(!).

But my point is: Only a pure theorist or someone desperately in need of another author credit would come up with these things. I happen to give rather good estimates for my projects, but when I look at how long the single tasks making up the project actually took, I have wild deviations in most cases. I think the problem is that in most cases, things go wrong or give problems no one ever even thought of (can we spell 'bug in compiler', API cannot be used as thought, etc.?). I agree that estimations get better throughout the project (I think estimations after 50% were done get pretty good - but on the other hand, it's not so much time left, so deviations tend to be smaller).

Isn't the main dilemma that we have software-developers on one side who would, in a perfect world and knowing how complex software development is, start developing and when an end is in sight start to even thing about a release date. And we have the businessmen in the companies on the other hand, who want to control _everything_ and have a fixed end-date so they can start using Excel to calculate or their fuzzy numbers. Any thoughts/opinions?

Joel On Software

"ask them how long it will take them to drive home this evening. When they reply "45 minutes," ask them "plus or minus what?" When they say "give or take 10 minutes," you got 'em :) If there's a 20% uncertainty on a task they've done hundreds of time with only a few unknowns, how can you do better on a novel task with possibly hundreds of unknowns" -- Former Fool

Estimating Time Accurately

How to use tool:

Accurate time estimation is a skill essential to good project management. It is important to get time estimates right for two main reasons:

1. Time estimates drive the setting of deadlines for delivery of projects, and hence peoples' assessments of your reliability
2. They often determine the pricing of contracts and hence their profitability.

Usually people vastly underestimate the amount of time needed to implement projects. This is true particularly when they are not familiar with the task to be carried out. They forget to take into account unexpected events or unscheduled high priority work. People also often simply fail to allow for the full complexity involved with a job.

This section discusses how to estimate time on small projects. Time estimates are important inputs into the other techniques used to organize and structure medium and large sized projects (Gantt charts and use of Critical Path Analysis). Both of these techniques reduce large projects down into a set of small projects.

Fully understanding the problem to solve

The first stage in estimating time accurately is to fully understand what you need to achieve. This involves reviewing the task in detail so that there are no unknowns. Inevitably it is the difficult-to-understand, tricky problems that take the greatest amount of time to solve.

The best way to review the job is to list all tasks in full detail. Simple techniques such as Drill-Down are useful for this.

Estimating time

You can only start to estimate time accurately when you have a detailed list of all the tasks that you must achieve. When you have this, you can make your best guess at how long each task will take to complete.

Ensure that within your estimate you also allow time for project management, detailed project planning, liaison with outside bodies, meetings, quality assurance and any supporting documentation necessary.

Also make sure that you have allowed time for:

- Other high urgency tasks to be carried out which will have priority over this one
- Accidents and emergencies
- Internal meetings
- Holidays and sickness in essential staff
- Contact with other customers, perhaps to arrange the next job
- Breakdowns in equipment
- Missed deliveries by suppliers
- Interruptions
- Quality control rejections
- Etc.

These factors may double (or more than double) the length of time needed to complete a project.

If the accuracy of time estimates is critical, you may find it effective to develop a systematic approach to including these factors. If possible, base this on past experience.

Key points:

You can lose a great deal of credibility by underestimating the length of time needed to implement a project. If you underestimate time, not only do you miss deadlines, you also put other project workers under unnecessary stress. Projects will become seriously unprofitable, and other tasks cannot be started.

The first step towards making good time estimates is to fully understand the problem to be solved.

You can then prepare a detailed list of tasks that must be achieved. This list should include all the administrative tasks and meetings you need to carry out as well as the work itself.

Finally, allow time for all the expected and unexpected disruptions and delays to work that will inevitably happen.

There are many, many ways to estimate a software project, ranging from formal models that can only be worked by a University professor through to a salesperson's hopeful guess.

Here we present a simple, proven software project estimation method that produces reasonably accurate results and has been used estimate substantial fixed price work.

This method is based on a quick but robust initial design on the premise that later versions of the design will reduce the amount of work as developers come up with smart ideas for saving work.

This method also calls for participation by a substantial number of people, increasing buy-in and gaining accuracy. It also follows the principles of the Cardboard Checklist for Planning, where relevant.

Overview

This estimation method has three steps:

1. Design the System.
2. Estimate each Part of the System.
3. Schedule the Work.

The outputs are:

1. A project schedule with allowances for schedule risk, internal dependencies and known outages.
2. A list of other assumptions made during the estimation.
3. A list of the project's external dependencies.
4. A list of other risks to the project schedule.

The inputs are such documents as are available, including:

1. The statement of project scope.
2. User requirements - use cases, functional specs.
3. Technical requirements.

Of these, the only one that is absolutely required is the Statement of Project Scope. However, thore more information available, the more accurate the final result will be.

Step 1: Design the System

Having assembled all the available requirements, produce an initial system design.

This is a design for estimation purposes only. Typically it combines the team's current thoughts on technical design with what was learned from carefully reviewing the known requirements. Assign at least one technical person with significant design skills to read and understand the requirement documents.

The design needs to be fleshed-out to to the subsystem level, with each subsystem representing 20 to 40 days work. Subsystems can be chunks of code that needs to be written, products that need to be configured or parts of both.

To attempt an initial design, it is necessary to have a statement of project scope. In addition to setting out a boundary for the system, the statement helps focus the development team on the delivery of useful functionality, rather than technical wizardry - something that tends to happen when the goal is unclear. It is not necessary to have all the final requirements, although it is helpful to have the core requirements clearly defined.

The initial design should be robust and complete - the team should have a high degree of confidence that the design will get the job done, even if it is not done in the most smartest or neatest way. Later revisions of the design, with the luxuries of time and more complete requirements can add technical elegance.

In order to get buy-in from team members, run a design workshop. Have some work done on design prior to the workshop by a respected senior designer, who then presents the results to the workshop for review. The workshop should take just a few hours - a half day at most, and be focused on identifying subsystems and checking whether the initial design will meet the user requirements.

After the workshop, the initial design should be written up as a formal project document. This document contains:

1. A system overview, with a diagram showing subsystems and their interactions with each other and with the outside world.
2. A description of the role and responsibilities of each of the subsystems.
3. A list of assumptions made in arriving at this design - including interpretations of the requirements.
4. A list of dependencies on external projects.
5. A list of other risks.

The lists of assumptions, external dependencies and risks form the basis of the final lists of assumptions, dependencies and risks.

Be sure to manually check that the design meets all known (and suspected) requirements.

Step 2: Estimate each Subsystem

With the design done and checked against the requirements, you are now in a position to run an estimation workshop.

Ideally, the workshop involves the individuals who will be doing the work. At least invite the project's team leaders and/or other senior technical people. For larger projects, you may want to divide the estimation into several workshops.

The estimation workshop should take one or two hours per subsystem.

The workshop has a simple format. Taking one subsystem at a time:

1. Break the implementation of the subsystem into tasks.
2. Estimate the likely time for each task.
3. Assign each task a 'schedule risk' of High, Medium or Low.
4. Document assumptions made.
5. Document dependencies on external projects.
6. Document any other risks identified.
7. After all substystems are estimated, estimate other project activities.

Examining each of these activities in more detail:

1. *Break the implementation of the subsystem into tasks.* Each task is a logical unit of work that will be completed by one person. Breaking the subsystem implementation into tasks will require an amount of on-the-spot design. Tasks should be three to seven days long and they include both coding and unit testing.

   There should be between four and twelve tasks for each subsystem.

2. *Estimate the likely time for each task.* Have the workshop participants figure out how long a task will take, based on their experience implementing similar things. In the absence of experience, make a bold guess and assign the task a 'High' schedule risk. (See below for a definition of schedule risk.)

   If developers are having trouble with the concept of 'likely', try this analogy: If the boss were to ring you up and say "Come to my office as soon as you are free", you might say, "Sure I'll be there in ten minutes". Now you may only take five minutes if the thing you were working on finishes more quickly than you thought. On the other hand, you may take an hour if you were to twist your ankle in the corridor. Ten minutes is the likely time, assuming nothing goes particularly well or particularly badly.

   Should you decide that a task is shorter than one day or more than ten days, consider combining or splitting tasks.

3. *Assign each task a 'schedule risk' of High, Medium or Low.* Some tasks are extremely straight forward, and have been done many times before, so it is possible to be very confident about the estimate given for the task. These tasks have a 'Low' schedule risk. Other tasks are almost completely unknown quantities, and these tasks have a 'High' schedule risk.

   This estimation method uses three categories of schedule risk as shown in the following table:

Category	Schedule Allowance
High	150%
Medium	50%
Low	10%

   A task with High risk can take 150% longer than estimated. In other words, nobody would be surprised if a 'High' risk task estimated to likely take five days to complete actually took a two and a half weeks.

   Similarly a three day 'Low' risk task should not take more than 3.3 days, and a four day 'Medium' risk task should not take more than six days.

4. *Document assumptions made.* List assumptions made in interpreting requirements. This list should build on the list started in the design phase.

   For example, it may not be clear what screen resolutions a system must support so the assumption will be made that "User workstations will have a screen resolution of at least 800x600".

5. *Document dependencies on external projects.* List outputs from other projects or teams that this project will depend upon.

   For example, if setting up a database schema will require the approval of the database admin team, write that down. This is a reminder to the project manager that they will need to secure the cooperation of that team.

   Another example of a dependency is where the project being estimated will use a component from another project that is not yet complete.

   It is also helpful for the workshop to estimate the chance that the dependency is not met and to document the effect of the dependency not being met. For instance, if that other project does not deliver the required component to a satisfactory standard, how must extra development will that cause?

6. *Document any other risks identified.* List any other factors that might affect the project. An example is the possibility that the test equipment may not be adequate.

Again, it is helpful for the workshop to estimate the chance that the risk event occurs and to document the effect on the project.

Do not document the risks that have been already been included into schedule risk, as that would be double-counting.

7. *Estimate other project activities.* The workshop will also need to estimate:
    - Additional design time required - for the whole system and for individual subsystems or components.
    - Prototyping time. Prototypes are often useful to reduce the total schedule risk for what would otherwise be long-running high-risk tasks.
    - Integration testing, System testing and User acceptance testing. Typically testing takes 20 to 30% of the estimated implementation time.
    - Set-up time, particularly if new equipment or development software will be used. This covers installation and initial configuration.
    - Development environment maintenance. Configuration management and build system administration fall into this category. Typically allow 10% of the estimated implementation time.
    - Release/Deployment time. Each release to the client takes time. If unsure, estimate one week for the entire team prior to each release for a combined integration test, fix and build.

Facilitate the workshop to ensure that it stays focused on the task of producing an accurate estimate:

- Initially, some developers may be wary of committing to any kind of estimate at all. Other developers may be over-optimistic in their estimates. Guide the workshop through this initial phase, emphasizing either that these are only estimates, with risks, assumptions and dependencies to be taken into account or that the entire project will be run according to their estimates, as appropriate.
- Detailed design discussion should not be allowed unless it is quickly resolved and dramatically affects the accuracy of the final estimate. Whenever a design discussion begins, encourage the participants to make a note and take it up after the meeting.
- Ensure that high-risk work is isolated into its own tasks, separate from low-risk work.
- In the event of several valid design options being presented to the workshop, choose the option with the least total time to implement, including risk allowance, and move on.
- If there is discussion about the meaning of a requirement, document an assumption about the meaning and move on.
- Every now and then there will be genuine disagreement between participants about how something should be done. In this case, get each of the warring parties to estimate their way, pick the estimate with the longest time to implement, including risk, and move on.

Collate the workshop output. Create a spreadsheet that contains one line for each task: the task name, 'likely' estimate, schedule risk assigned and total (likely plus schedule risk).

To these numbers, add an allowance for risks and external dependencies identified. Assuming the risks identified are relatively small and are independent of one another, calculate the allowance by multiplying the percentage chance of the risk occurring by the cost (in $ or days) to fix it.

Large risks - those that can stop the whole project will have to be dealt with separately in the project plan. Risk that are not independent - those that will probably occur together if they do occur should be aggregated into larger risks.

Finally add an amount for project management. This will range anywhere from 10% of the estimated development time upwards, depending on the overall project risk. Include in this an allowance for change management and contract negotiation. If the time estimated to manage the project exceeds the amount of time the project manager will have available, consider asking for extra resources such as a project administration assistant or a full-time client manager.

Congratulations! You now have all the numbers that you need to provide senior management with a fairly detailed project budget, including estimates of delivery dates.

Step 3: Schedule the Work

Before promising a particular delivery date, you will need to schedule the work. This also provides an excellent opportunity to get the whole development team involved in the project and committed to delivery dates.

<advertisement>

If Cardboard Schedule were still being sold, this is where the advertisement for it would go.

</advertisement>

Having said that, the principles involved are reasonably generic and you may find other tools suitable. However, do try to steer away from Gantt chart software, as the chart quickly becomes too large and complex for scheduling in a workshop.

Prior to the workshop, create a draft schedule:

- If you have not already, work out how many people will be needed on the team in order to complete it in the required time. Do this roughly be dividing the total person days by the available working days. Add these people to the schedule.
- Add project milestones, including interim and final release dates.
- Add each of the tasks from Step 2 above. Make each task the length of the 'likely' estimate.
- Assign tasks as best you can without consulting individual team members. Distribute with blank space between to represent schedule risk.
- Add dummy tasks to represent known outages - public holidays, annual leave, training courses and so on.

On a larger project, it might be necessary to split the work amongst several teams, each with their own schedule. In this case, use milestones to co-ordinate the schedules.

All team members whose work is being scheduled should participate in the scheduling workshop. Ensure that at least some of the participants from the estimation workshop are there as well, to explain and clarify as required.

The workshop should be kept short and to the point. If the participants are familiar with the task based estimates, and the draft schedule is reasonably complete, allow five minutes for each eight person weeks work to be scheduled. Allow more time for an overview and explanation of the estimate if necessary.

The format of the workshop is simple. For each logical grouping of tasks:

- Show the relevant portion of the schedule to the workshop on a large monitor. The group looks at the schedule and makes suggestions. Make changes to the schedule on-the-spot.

- Ensure that tasks are assigned to people who have the skills needed to do the task.
- The person who will be doing each task should be given an opportunity to express an opinion about the task's estimate.
- Try to move high risk tasks as near to the beginning of the schedule as possible so that there is as much time as possible to deal with problems.
- Ensure that dependencies between tasks are respected - reorganize the schedule if necessary.
- Ensure that no-one is assigned too many tasks or too few. There should be some gaps between tasks to represent schedule risk.

At the conclusion of the workshop, you will have a schedule for your project that accurately reflects your estimates. Keep in mind however, that the schedule will change (hopefully for the better) as the result of further design changes.

Observations

- While this method is presented as a series of steps, don't be totally limited by this sequence. For instance, it is valid to change the design while workshoping the estimate. Try to limit backwards jumps though, as they can be costly.
- This is just one method for obtaining an estimate. While it is suitable to be used as the primary method of estimation, do check the answers through other methods such as comparing it with similar projects or counting function points.
- Schedules arrived at with group commitment tend to be self fulfilling. If possible, ensure that every developer at least attends the scheduling workshop.
- Like many estimation methods, this one relies on experience. After several projects a team should be able to estimate projects to within 10%. Significant improvement in accuracy can be made over time by the collection and use of suitable metrics. However, do be aware that the working with unfamiliar technology or in new business areas necessarily makes estimates based on experience less accurate.
- Finally, remember that a schedule is of most use when the project is tracked against the schedule. Keep the schedule up-to-date with latest design changes. Also, check-off progress against the schedule regularly, perhaps in a weekly team meeting.