How long should a software project take?

A collection of Emails and responses published with permission
Assembled by R. Max Wideman on http://www.maxwideman.com

Introduction

From time to time I get inquires that arouse my interest. Here is one that I think is a very legitimate question and first I try to answer from a purely pragmatic project management perspective. Later in this article you'll find responses from experts in the field.

Anand N. wrote by Email 8/7/05

Hi

I have been following your project management site for a while now and it has been extremely useful. I'm in the position of Software Application Manager with the technology department of a Financial Services major and am based in Singapore.

Due to the nature of competition in the industry segment I work in, projects we work on are extremely tight as the business and management find the need to get the product to market as quick as possible and this "time to market" factor invariably impacts the end quality of the project leading to post implementation issues etc. Our projects are kicked off with the schedule fixed - the effort very rarely drives the schedule leading to a lot of deficiencies. Management understands these factors but we just seem to have to live with it.

One of the lines I use with management to fight for a better project schedule or duration window is: "A 100 man-day project cannot be practically completed in 1 day with 100 resources. Neither is it practical to complete it in 2 days with 50 resources - though possible on paper in an ideal situation.

My question to you is: "Are there any tools/formulae to estimate the proper and practical schedule for a software development project, given that the total effort is known?"

Any guidelines you have worked with or pointers to where I can refer will be appreciated.

Thanks, Anand.

Max's thoughts on the issue

Dear Anand:

You pose an interesting question. For all the "theory" and tools and techniques that we like to discuss in project management, when it comes down to it, "real" projects turn out to be quite different!

The short answer to your question is that there are so many variables involved that there is no simple calculation. However, if you are able to make sufficient assumptions, then you should be able to come up with some sort of rule-of-thumb that reflects your particular environment. I'll try and answer your question from "first principles" – but to do so, I'll have to make a lot of those assumptions. However, I have also consulted some of my friends who have provided more experienced input than I can that I'll share with you in a moment.

I'll first paraphrase your question as follows: "What should be the proper length of a practical schedule for a software development project of 100 man-days of effort?" That's about 1000 person-hours including unpaid overtime, or about $100,000 gross. That's not a large project but a nice one for a lot of people.

The first point to note is that any statement of effort before a project starts is an estimate and we can never be sure of the accuracy and reliability of that estimate. In this case, one thing we can almost be sure of is that it will not be exactly 100 man-days. It might be less or it might be more but the likelihood of it being exactly 100 is quite remote. Moreover, there is a much higher probability that it will be more rather than less because it is much more likely that we have overlooked part of the work rather than including work that will not be necessary.

There is a simple lesson here. If you are preparing the estimate, and have the opportunity to do so, then add an extra activity, preferably at the end, that could, if the worst comes to the worst, be omitted. Personally, I always add an activity such as "Final documentation and cleanup" and I try to make that about 3% of the total schedule. It's not a lot, but it is difficult for management to argue against and it is better than nothing.

The next issue is that if we do indeed know, with some confidence, the amount of effort involved, some work has already been done in formulating the project, objectives, and requirements, that sort of stuff. Consequently, we are presumably now talking about the project execution phase. Or, in software parlance, we are talking about "construction" and, further, let us assume that the effort involved in the final "transition" phase is not included in the 100 man-days. If it is, then back those man-days out of the total for the reason I'll explain in a moment.

The next question is: What does the project look like? Does it have elements that can be separated as in a work breakdown structure? How are those related and how many elements can be worked on concurrently? It is rare that you can work on all elements at once - there is usually some sort of precedence involved, a critical path.

But the real nub of the matter is how many people can work satisfactorily together as a team on any one of those elements? That appears to be highly dependent upon two factors. The first factor is the technology methodology you use, and the second is the make up of the team.

On the question of methodology, if you use pure waterfall, you may appear to get it done faster but the risks are much higher, certainly the risks of unfound coding errors, if not outright failure. And those errors found late in the work lead to much longer delays and higher costs to fix. If, however, you use a methodology such as the Rational Unified Process ("RUP") approach, the elapsed time may be a little longer but only because you will be consuming your resources more prudently and effectively. And further, your probability of a successful product will be higher.

On the question of the team makeup, my next assumption is that you have assembled the right mix of roles for the project. After that, the results are highly dependent on the quality of the team members, how experienced they are, how well they know each other's work habits, and how comfortable they are with each other. Still, we can hazard a guess. Conventional wisdom, i.e. practical experience, has it that the optimum number in a working group, including the leader, ranges from five to nine.

Let us pick the average number seven encompassing the roles of, say, architect/team-lead, analyst, developer, and tester. Let us also assume that these individuals can perform more than one role in each case. This will enable better distribution of the work as the work profile varies, i.e. internal "resource leveling".

Then theoretically it would take about fifteen days for this group to consume the 100 man-days. But we know that, in spite of resource leveling, no effort on a task can be mounted at 100% capacity to start of with, nor can it work flat out and stop equally suddenly. You simply have to "ramp up" as more work becomes accessible and "ramp down" as the work peters out. The profile of this effort curve is itself controversial.

For the efficient consumption of our estimated budget hours we must therefore also assume that the number of qualified team people available is variable. That is, we can bring them on and send them off to match exactly the work availability or accessibility. This assumption is also controversial since managements typically don't like to see people standing idle while they are "waiting to be engaged", but we'll let that pass.

I have discussed "Resource Loading" in my paper "Applying Resource Loading, Production & Learning Curves to Construction: A Pragmatic Approach". You will find it here: http://www.maxwideman.com/papers/resource/abstract.htm. In particular, you will find a typical profile in "Figure 6: Histogram, envelope, and empirical resource loading input" here: http://www.maxwideman.com/papers/resource/learned.htm#fig6. True this is taken from the building industry, but for any significant sustained efforts the principle probably holds true.

The reason why I use the "construction" stage of a project as the yardstick is because it is the only part of a project that is sufficiently tangible to have an intrinsic effort content that can be estimated. All the other phases and stages are as long as, or short as, you want to make them – for better or worse!

My Figure 6 shows that 100% engagement of the full team lasts for only 25% of the total elapsed time. Assuming this to be true for software development (another big assumption) then we can calculate that the "best" duration of the "project" under these assumptions is around 23 days. Note, however, that this number is very sensitive to the percentage duration of the *full engagement* of the *full team*. Note also how I have expressed this, because it is often very difficult to measure in practice. The *full team* may be charging time to the project for a greater proportion of the time but that does not necessarily mean that they are *fully engaged* for all of that time!

So, the 23 days should be possible, but I suspect that it, too, is optimistic because it does not allow for:

a. Any delays arising from required interaction with the "stakeholders"
b. The restraining effect of the number of iterations required for the work in question

Still, if my assumptions are anywhere close for your case, you might establish a Rule-of-Thumb of around 25% of the man-day effort estimate for the project duration. If your teams are smaller (or larger) then you will need to recalculate accordingly, and adjust for any other of my assumptions that do not reflect your situation.

So, after all that, here are some real expert opinions on the next pages.

Joe Marasco comments by Email 8/8/05

Joe Marasco is a retired Senior Vice President with Rational Software, now a part of IBM. He spends his time indulging in many hobbies, including writing. You can find out more about him on his web site at http://www.barbecuejoe.com

Wow. This is a "trick question." Instead of making detailed comments on your document, let me say the following.

The problem is that ideally we would like to do all projects with one small team, say seven people, to use your example. However, a 100 "man-day" project is an extremely small one in the software world. Many software projects are just too big to be done by seven people.

One should always assume an S curve for the resources applied and time to completion. Note that the relatively flat ramps at the beginning and the end will tend to make the project stretch out, but there is not much you can do about those. They represent the fixed overheads of getting started and completing all the messy details at the end.

On the other hand, if you have done a good job during inception and elaboration, you can hope to optimize during construction, the ramp part of the project.

But be very careful about removing inception, elaboration, and transition phases from the estimate, leaving only "construction." The fallacy here is that these activities take significant time. In the old days people used to do this all the time; they would estimate "three weeks" for a job, meaning that they could write the actual code in three weeks. This corresponds to three weeks of what you might roughly call construction; in that model, the actual time to completion from start to finish of the project could easily be several times that. It took a long time for people to be broken of that habit. Note also that sometimes the coding effort is the easiest to estimate, whilst the other activities have variability that is much larger and predictability that is much lower. So an estimate of the "coding" or "construction phase" (please note that these are NOT the same!) is usually a poor predictor of how long the entire effort will take.

There is no one magic formula, no one recipe. On the other hand, starting with 100 man-days as a typical project may lead you to some spurious conclusions. Why don't you ask your correspondent for his typical sized project, and then work backwards from there? I know he used 100 man-days in his hypothetical example to his management, but is that the real size and scope of the projects he deals with? If the answer is that there is no "typical project," then ask him for what constitutes a small, medium, and large project in his world. Then you can come up with three guiding scenarios. Any "one size fits all" approach is destined to fail in this arena.

I think this fellow would profit from reading my book, or at least sections of it. If he hasn't read *The Mythical Man-Month*, he should read that one first.

Thanks, Joe

Joe adds later

Compared to longer, even multiyear efforts, relatively quick-hit projects are a problem too, because the smaller the estimate the higher the intrinsic risk, relatively speaking.

Max's comments

That looks like sound advice. The books Joe mentions are:

- *The Mythical Man-Month* by Frederick P. Brooks, Jr., Addison Wesley, 1995
- *The Software Development Edge* by Joe Marasco*,* Addison Wesley, 2005

Joe's book contains some delightful anecdotes and wisdom by a mythical hardball pragmatist known as Roscoe Leroy. In particular, after you have figured out how long your project is going to take, you can use Roscoe's "Square root rule" to figure out how late your project will probably be (in Chapter 11 – Scheduling). By the way, the trapezium that I have adopted is a close approximation to the "S curve" that Joe mentions. The trapezium greatly simplifies the mathematics and I am all for simplicity.

Bob Steinberger comments by Email 8/8/05

Hi Max,

You provoked me to reply and in so doing, I am sure to provoke others to come forward as well. Hopefully, you will accomplish your desires.

Here goes. On the issue of optimal number of people on a project: Since you brought in some fine examples of progress curves from the construction industry, let's go one step beyond.

Suppose you were to build a pilot plant to manufacture diesel oil from coal. Since a project of this type involves different types of project teams, would 7 of any team type be optimal? I guess you could decompose tasks down into working teams and use 7 as optimal. But that is pushing the idea of 7 a bit hard since there would be many groups of 7 that make up the work force.

One the other hand, Moses was a team of one, though some might say he was not operating in an optimal mode.

I think the word optimal depends upon what it is you wish to optimize. For design projects, it is customary for an engineering office to perform design work in Location A using shift A workers and work a second shift at location B across the world using shift B workers, accomplishing 2x amount of work per day and shortening the design calendar time. The optimal team size might be one or two designers and/or a working manager.

For Middle East construction projects, a 60-hour, 3-shift week is quite common. Getting down to individual crew sizes, while seven might be neat in the US, other locations that use low cost labor tend to staff crews with more people because of the politics of acquiring a workforce or because of local mandates on crew sizes.

If you have a chance to watch a new building under construction, I think you will be amazed at the small number of workers at the site. For a given task, you might find one equipment operator and perhaps one or two others in the operator's team. People pushing a wheelbarrow in the US have given way to automated equipment, including such gear as concrete pumps. Elsewhere, heavy equipment may be considered elephants and light equipment being thousands of day laborers using their hands.

But then one might argue that none of this has anything to do with software development. But you can piece all of this together if you consider the workforce, their tools, and what they would otherwise do if not employed.

From my point of view, I do develop software and am optimal as a team of one.

Bob

Max's comments

That last is an interesting comment. Some personalities just love the idea of teamwork, they thrive on it. Others just can't stand it. It all depends on whether you are a "process oriented person" (let's keep it going) or a "product oriented person" (let's get it done.) Both possibilities must be factored into the particular project team environment you are working with. Anyway, "Bob" is Dr. Robert L. Steinberger, Technology Fellow Integrated Engineering at Aspen Technology, Inc. at the Icarus Office in Gaithersburg, MD.

Philippe Kruchten comments by Email 8/8/05

Max,

The relationship between total effort and project duration is one of the best understood in software engineering, mostly through the work of Barry Boehm and the COCOMO. A few cost drivers affect the result, as well as different calibration depending on the type of software (MIS application, embedded, etc.). COCOMO was developed in 1981, and its successor COCOMO II in 2000.

For example: A 100 person-month project, with all cost drivers at nominal value takes 14 months according to COCOMO; staffing would start at 3 persons, and culminate at 8 or 9 (7.1 average) in what we call the construction phase in RUP. But a 200 person-month project would take 19 months with staffing going from 5 to 12 (10.5 average). It would however deliver less than twice the code (35K for the first 67K for the 2nd).

There are numerous COCOMO resources on the net. I often use the NASA calculator: http://www1.jsc.nasa.gov/bu2/COCOMO.html. The same page contains plenty of other pointers.

Not exact science, but allows you to confirm some "Leroy Roscoe" rule of thumb numbers, like: Are you in the right ballpark, or off by a factor 2 or 3 . . .

Cheers,

Philippe

Philippe adds later:

Here are the COCOMO 1 cost drivers I referred to:

1. Product attributes
   a. Required software reliability
   b. Size of application database
   c. Complexity of the product

2. Hardware attributes
    a. Run-time performance constraints
    b. Memory constraints
    c. Volatility of the virtual machine environment
    d. Required turnaround time

3. Personnel attributes
    a. Analyst capability
    b. Software engineer capability
    c. Applications experience
    d. Virtual machine experience
    e. Programming language experience

4. Project attributes
    a. Use of software tools
    b. Application of software engineering methods
    c. Required development schedule

For an explanation see:
http://www.mhhe.com/engcs/compsci/pressman/information/olc/COCOMO.html
For COCOMO 2 see: http://sunset.usc.edu/research/COCOMOII/expert_cocomo/drivers.html
For more on cost drivers see: http://www.softstarsystems.com/cdtable.htm (COSTAR is a commercial tool that uses and extends COCOMO)

However, 100 man-day projects are probably too small for COCOMO. A 100 man-day project is probably best done with 3 people x 6 weeks, or 4 persons x 5 weeks. After that you have a hard time distributing the work and synchronizing. There are exceptions to this, for example:

- If there are many different and simple things to do, requiring different expertise, and
- When the architecture of the software system is stable, understood and untouched as in the case of an update of an existing system. (All these things are taken into account by COCOMO cost drivers.)

Max's comments

The size of project that Philippe quotes for applying COCOMO is, of course, much larger than we originally postulated at the beginning. Perhaps the concept of "economies of scale" is at work here? Certainly for small projects, the smaller the project the higher is the percentage of administrative overhead. Even so, the resulting durations shown by COCOMO seem to me to be quite optimistic. Perhaps these are the "theoretical" durations before the application of allowances for the "cost drivers" that, in practice, inevitably take over. On this basis the number of software development projects that end up being late should not surprise us. The problem is not with the project but with the baseline of reference!

Philippe's later observations for a 100 man-day project seem much more realistic and consistent with our own predictions. Philippe is Dr. Philippe Kruchten, Professor of Electrical Engineering, University of British Columbia, BC, formerly lead architect for the Rational Unified Process. His latest book is *The Rational Unified Process Made Easy – A Practitioner's Guide to the RUP*, Addison Wesley, 2003

Gary Pollice comments by Email 8/8/05

Philippe and Bob have provided some interesting fodder for the discussion.

COCOMO is certainly one of the best tools we have for predictive metrics, but even so, it is not very reliable. Some estimates show it to be quite variable compared to many other engineering disciplines. This, of course, begs the question as to what software engineering really is, but that's for another discussion. I think Philippe has it right in his last paragraph:

> "Not exact science, but allows you to confirm some "Leroy Roscoe" rule of thumb numbers, like: Are you in the right ballpark, or off by a factor 2 or 3 . . . "

Is there an optimal number for any project? Sure, but we probably can't determine that. Besides the cost drivers, you have to consider the type of people, how well or poorly they work together, size of project, etc. There are many people who are quite optimal when working on large projects, but don't do so well on smaller teams. Figuring out the optimal time for a project, given a certain number of "specific" people might be possible with some accuracy as long as you've got enough historical information about how the people have worked together in the past.

If you are using some sort of iterative development, you can get better and better estimates for the next iteration or two by using what has happened before. Optimal estimates for a large project that will take a lot of time -- I'm fairly convinced that we're a long way from that. The COCOMO database has a lot of data in it, but it just scratches the surface of what we need to know to get better at believable estimation for large, multi-year projects, IMHO.

Gary

Gary adds later

Where relatively quick-hit projects are concerned, this is where I think some of the agile folks have something to say. Quick iterations, continual focus on the most important requirements (determined by the customer) and about all they can guarantee is that they will produce X hours of work a week, but if you select the right pieces, you get the most important X hours worth of value.

This clearly doesn't scale up very well.

Max's comments

So there you have it! A lot of possibilities for estimating project durations but a lot of things to take into account in practice. The best you can do is to collect data from your own environment and convert it to a form that you can use to predict future project durations. Then, if you want to improve on that, look for the "management obstacles" that retard progress and get them removed!

What are "management obstacles"? Anything that causes stress to the working team resulting in reduced productivity and quality of work. Research indicates that the top five are:[1]

1. Constant interruptions
2. Unreasonable deadlines
3. Poor internal communications

4. Lack of support
5. Incompetent senior management

By the way, Gary Pollice is Professor of Practice, Computer Science, at Worcester Polytechnic Institute Worcester, MA.

Keerin Saeed forwards advice from Steve Cotterell by Email 9/1/05 from across the UK

Dear Max,

Thank you for your Ask an Expert question.

You asked, "How long should a software, or IS/IT project take?"

Steve Cotterell, Technical Editor of Project Manager Today replied:

Because of their nature, I think that 'normal' commercial software projects lend themselves to treatment as RAD (Rapid Application Development) projects. Basically this involves setting a time limit on the development, prioritizing the work to be done and accepting that, on the agreed end date, the project is finished.

The opposite of this method is to treat a software development as a normal project, accepting changes in specification etc. as the project evolves and allowing the developers to tinker with each piece of code until they consider it to be 'perfect' (and it never will be!). The danger with this approach is that the project is certain to over run, wildly exceed its budget and, as a result, fail.

The need is, therefore, to keep the development team focused, with a sense of urgency and, I suggest, this means that the time to finish should be measured in weeks or (a few) months - no more than, say, five or six. If this is not going to be possible, because of the sheer size of the task, then I suggest that, before it is presented to development teams, it should be broken down into separate applications, each of a more manageable size, and treated as a programme of projects, each project handling a separate application. These projects should be worked on, concurrently, by different teams. There should, of course, be inter-team and inter project liaison!

This is even more important if the specifications of the job to be done are affected by something that is subject to rapid change, without much notice - government legislation comes immediately to mind. If the legislation that the software is designed to comply with changes during its development, the project is almost certain to run into difficulties.

Having issued a successful first version of the application, there is no reason at all why another, similar project should not be set up to produce version two which incorporates some of the functionality that there was not time to include in version one, fixes bugs (there are always bugs) and incorporates (if possible) any changes in specification that have arisen since the specification for version one was agreed.

Software projects that involve safety and/or matters of life and death (for example, air traffic control) are a different matter. Whilst it's still important to keep the developers focused and with the same sense of urgency, there has to be a much greater emphasis on accuracy and testing and so the additional cost of providing these will have to be written into the project budgets from the outset.

I hope this answers your question

Kind Regards,

Keerin
keerin.saeed@apmgroup.co.uk

Max's comments

Steve Cotterell is Technical Editor of Project Manager Today and wishes to retain copyright to his response so that neither it, nor extracts from it, may be re-used elsewhere without his express written permission. Project Manager Today is a print magazine and membership-based site supporting the project management community in the United Kingdom. Keerin Saeed is Webmaster of The APM Group's Interactive Community of Practice (www.apmg-icp.com), the first online community for anyone involved in program, project and risk management.

[1] Wheatley, R., *Taking the Strain*, Institute of Management, UK, 2000