

The Art of the Schedule

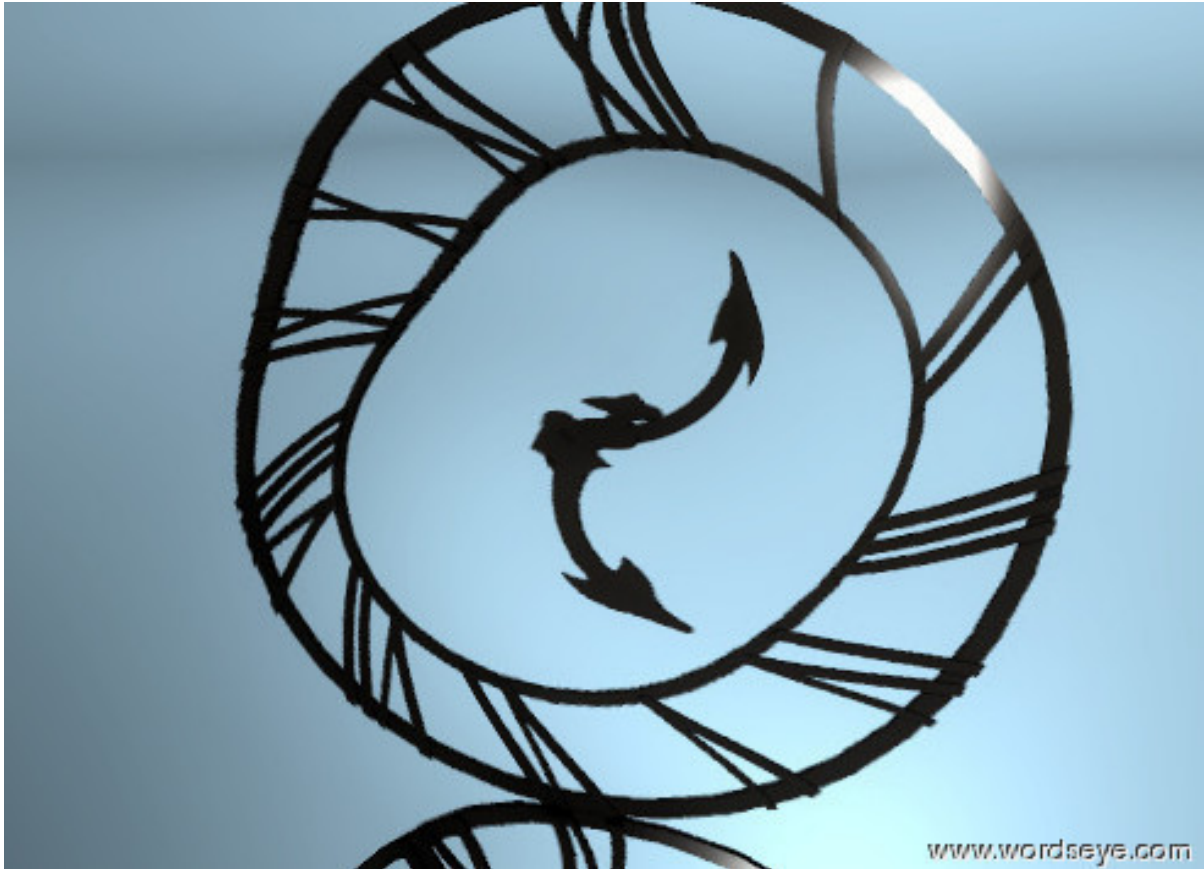
by Philip Chu

Table of contents

1 Publication Information.....	2
2 What is a Schedule?.....	2
3 It's a Deadline, Not a Guideline.....	3
4 It's Macro, Not Micro.....	4
5 It's Not a Negotiation.....	5
6 Schedule Suboptimally.....	6
7 The First Schedule is Always Right.....	6

1. Publication Information

Copyright ©2004-2008 by Philip Chu All rights reserved.



"A canna' change the laws of physics. I've got to have thirty minutes." - Scotty, in *The Naked Time*

2. What is a Schedule?

On many projects I've been told there is no schedule - either as a complaint ("We don't have a schedule!") or boast ("We don't have a schedule!"). But there is always a schedule, and you'll find out what it was when you don't meet it.

- During an interview with a prospective client, I was told "we're not going to release this product until it's ready." Soon after I started working there, the president of the company called an all-hands "come to Jesus" company meeting, where he yelled at everyone for

not having the product finished several months ago.

"We won't release the product until it's ready" - famous last words. Only well-funded and self-funded companies can really say it, mean it, and do it, and you usually hear this said after a product misses its original release date. So how do you make schedules you can meet and meet the schedules that you make?

Before blathering on about effective project scheduling, let's be clear about the definition of a schedule. A schedule is an expectation of delivery dates - most importantly the product release, but also key demos, milestone deliveries to customers, promised interim release dates. Don't confuse the schedule with a glorified task list, a daily to-do list, or lines of code written per day.

- The management of a startup company that was cruising in research and development was obviously getting restless, so I suggested we establish a schedule. This inspired the project manager to establish a day-to-day task list for each engineer during the next two weeks. When I clarified that I really meant "when do we have to have something you're going to show", upper management commiserated and then informed us that we had to show a demo to investors at the company launch party in a few weeks! The micromanagement list went out the window and we scrambled to implement the key features needed for demo. (We came close, but I can't say it was a success - we got the demo running just as the party was winding down, and the funding fell through.)

A project's life or death hinges on getting good demos in the premier trade shows, satisfying contractual deliveries to clients, and timely appearance on the market. The entire project revolves around these expectations, so the first job in scheduling is to figure out these expectations and make them and their assumptions explicit.

3. It's a Deadline, Not a Guideline

After understanding what a schedule is, the next important thing is to take it seriously. Know your schedule, and make sure everyone on your project knows it.

- I was astounded on one project when casually mentioned our recent product release and a fellow engineer on the project didn't know what I was talking about. But I couldn't blame him for that head-in-the-sand mentality - our management would announce "stop coding, we're making a release today" and then justify it with "well, the beta release date was last week."

Any computer scientist should be able to tell you that constraints are a good thing when it comes to problem-solving - they narrow down what you can and can't do. Take advantage of the fixed milestones in your schedule, like trade shows. Milestones that you can't shuffle

around in self-denial are useful, like the scale that won't lie about your weight.

Every missed deadline means the next one will be taken that much less seriously. And every project that runs overtime means the next project will start and end that much later.

- In a surreal moment, just a few weeks before a product release, timed to coincide with the premier industry trade show, and right after I'd dissuaded the company president from demanding new features by pointing at the bug list, a well-intentioned marketing manager suggested to me "why don't we postpone the release by a month so you guys have more time to work on it?" Setting aside the trade show, which should have concerned him more than me, we had a supplementary feature release scheduled just three months after that, and then big upgrade a few months later. Delaying the release by a month would have just delayed all the other releases. And it wouldn't have saved us any work - we would just have been behind by one release month.

Stick with the schedule. There's always a next time, if you get this one done on time. And while delivering late is criminal, there's no sin in delivering early.

- For some reason, even when given the option to either a) release the product early or b) take the extra time to test the product, management usually elects option c) squeeze in a new feature that jeopardizes the the product and the schedule. (In fairness, engineers are often complicit - "sure, it'll just take an afternoon"). The one time I've seen a product delivered early took place after the president of the company gave the development manager a fake release date. I can't imagine that working more than once, though.

4. It's Macro, Not Micro

When I shop for groceries, I know it's not going to take more than half an hour, and I'll spend about forty dollars (sixty, if I'm hungry). I know this because I've bought groceries before, and that's generally how long and how much it takes.

Now, if I plan my grocery expedition by listing all the items I think I'm going to buy, estimating the unit price of each item, and predicting how much time it will take to park the car, locate each item, stand in the checkout line, and unload each item, my schedule will be off by the time I reach the first aisle. It's unlikely that my predicted individual prices will be on the money, so to speak, but it would be a waste of time to recalculate my schedule every time I find an item and its actual price. Even if I have a detailed grocery list written down, I'm only committed to the items I really need (can't postpone getting that cat litter) - I may purchase additional items that happen to be on sale and omit items that turn out to be too expensive or are unavailable.

The same goes for project scheduling. Estimate project durations based on what you know of

previous projects - that's where experience comes in, and that's a big reason why senior engineers should get paid more, if they've learned anything. Estimating project durations by summing up individual tasks only works if you've worked on exactly the same kind of project before and know every single task and possible schedule deviations that could happen in that type of project. In which case, you'd already know how long the total thing took, so why not use that, anyway?

- I worked on one distributed simulation project where the manager's office wall was completely covered with a Microsoft Project schedule. I assume this included every task going into the project (and maybe tasks not going in). But this project actually had a very simple schedule - have the simulation ready for the exercise date in six months, or the exercise would proceed without us. Major milestones, like having the software components ready for integration, the network hardware up and running comfortably ahead of the exercise date, and even my security clearance so I could start work, took months longer than they should have. Yet we had our meetings where everyone reported that everything was going fine and more paper went on that scheduling wall. The final month of that project was an exercise in panic and triage.

Schedule the major milestones first - the final release, the beta and alpha releases, and the interim milestones like first QA release and trade show demos. These are your hard targets - everything else can, and will, be more malleable, as you find certain technologies are working out better than others, you have to deal with staff turnover, sick days and vacations. I personally don't feel it's useful to schedule tasks to a granularity smaller than a week, but regardless, don't let your micro-schedule drive your macro-schedule - tracking low-level tasks looks good on paper, but the feeling of control is illusory - it's no substitute for "macro" planning.

5. It's Not a Negotiation

I've worked with managers and clients who would ask for a time estimate on a task, and then respond to the estimate with "why should it take so long", or "can you get it done sooner?". There are several things wrong with this approach:

- If you're taking issue with an estimate, that indicates you already had a preconception of what the estimate would be.
- There's no reason to believe a revised estimate will be more reliable than the first one. On the contrary - it will probably be less valid.

Remember what a schedule is - a prediction. I have allowed myself in some cases to lowering time estimates in these "negotiations", but it didn't change how much time the task actually required. If, as a manager, you are asking for a lower time estimate, then you are saying the

original estimate is wrong, and if you receive a revised prediction, whether it's more or less to your liking, you should ask yourself why the revision is more believable than the original prediction.

Likewise, as a client, you may want to bargain down on price, but bargaining down the time estimate will just backfire on you - the project will take as long as it takes.

That doesn't mean you cannot shorten the time for a project, but that means making reasoned tradeoff in features or resources. The right question to ask is "What can we do to get this project done earlier"

6. Schedule Suboptimally

I used to have a terrible punctuality problem when I lived in Boston. Whenever an appointment time came around, I would tell myself I just need, say, fifteen minutes to drive down Memorial Drive along the Charles River to make a meeting around MIT. Invariably, after giving myself that fifteen minutes, I would still get started late, get bogged down in Boston traffic (sometimes I forgot to factor in the rush hour impedance). And then there was the time spent getting lost, if it was an unfamiliar meeting place, finding parking (certainly a high risk factor in Boston), and navigating the destination office building.

This planning could be charitably called "best-case" planning, but I think everyone would agree it's a stupid way to schedule, especially after missing more than one meeting (including job interviews) by a wide margin. Nowadays, I'm much better at this - partly because I moved to California, but also because I do take in account how long these trips usually take, and I give myself some extra time in case traffic is slow or I have to stop for gas.

Software projects should be scheduled the same way. Look at how long it's taken your group to do similar projects and add some safety time to take in account things that might happen. You don't necessarily have to take in account worst-case scenarios - I don't factor in potential three-hour freeway stoppages when I plan my outings, but they do happen once in a while. But do consider carefully how much risk is acceptable, a question that is de rigeur in other aspects of engineering.

7. The First Schedule is Always Right

One consequence of making a distinction between the macro schedule and micro schedule is sticking to the initial macro schedule. The schedule you set down at the beginning of a project is most likely the correct schedule - you're basing the delivery date on market considerations, expected budget, staffing and technology capability, and the beta, alpha and interim milestones all fall from that. At this point, you're most free of wishful thinking influences - the schedule should be consistent with your experience from your own projects

and others that this is a feasible schedule. Some things will take longer than expected, some shorter, but you'll arrive at the expected time.

- Three months into a a one year development schedule to port a 3D graphics application from Unix to Windows, I suffered several bouts of unwarranted optimism. The user interface system was basically up and running, and I told my boss I thought at this rate, I'd have the project done six months earlier. Fortunately, he didn't tell anyone. With Windows idiosyncracies, graphics card issues, tracking ongoing changes from the Unix product line, and battling a compiler that was also in a "pre-alpha" stage, it wasn't long before it was obvious we were exactly where the originally schedule said we'd be.

But no good deed goes unpunished. Once it appears your project won't be a disaster, everyone who was keeping a healthy distance away will suddenly show up to "help".

- Ten months into a year-long crunch project, at the beginning of which it was wisely decided to pare features to enable a release by the next industry show, my boss suddenly wanted to put those features back in. He went around me (he knew what I would say) and asked some of the more optimistic and eager-to-please engineers to put in the features. After several days effort, I put my foot down and explained we already had a hundred and fifty bugs marked by QA as critical, and we could not afford the time to put in new features. "Oh, I didn't know we had so many bugs," was his lame acquiescence.
- On a game project at a different company, the president tried to accerate the project by promising game delivery two months ahead of schedule. It seemed like things were going well, and finishing this project early would have let us get started on the next game, so I agreed, but once again, I was wrong. When we reached the revised deadline, we were just starting find all the bugs that had to be fixed, including extensive requirements by the console makers, whose approval we needed to distribute the game.

So we went back to the original schedule, but during the last week, once again, the president tried to advance the submission date by a couple of days. And once again, we didn't make it. Worse, for every attempt to finish early, there was an extra flurry of activity, extra builds and late nights, that were essentially wasted our energy and sapped the momentum of the team. When you're taking catnaps in your parked car during the day, you're probably not producing quality work (especially if you own a compact car)

There will always be at least one point in the project where it seems like things are going better than expected. It's not. If the project actually gets done early (but it won't), then ship it, and if you want to maximize the chances of that happening (but it won't), then focus on getting those requirements done and avoid other distractions, like additional features.