On Schedules

First lesson

The first project schedule I created was at Frame Technology (1992-1995). I had accepted an offer to manage a team of qa engineers responsible for testing the Macintosh version of FrameMaker[i]. With the job came the usual administrative tasks, including creating schedules for various projects.

Like many managers in smaller software companies, I was promoted but given no formal training in management, let alone any guidance on creating a schedule. At my first team meeting as a manager, the marketing manager (who also ran the project) asked me how long a particular set of tasks would take. As well as being a new manager, I was somewhat new to the Macintosh platform, having worked in Windows/DOS for most of my career. So when the product manager awaited my response, I told her I'd have to get back to her. She paused, knowing my situation as a new manager, then asked when did I think I would have a schedule?

That's when I got my first lesson on crafting a schedule. If you aren't ready to provide a schedule, that's not necessarily a bad thing. But, you should, must, promptly commit to when you will have a schedule. This is sometimes called giving a date for a date. If there's one schedule related lesson I impress on managers and senior people working for me, it's to never, ever say "I don't know how long it will take" and then not follow up with "But I'll know by the end of the week." Do this in my department and you should start to update your resume.

Questions and more questions

After that first meeting I talked to some of the more experienced software managers. Most of them, I learned, had no formal training in scheduling, but had been creating schedules for a number of years. I started a list of questions which, like chisel strokes on the uncarved block, work away to reveal a schedule. Over the years the list has grown. Note that some of these questions are conditional, depending on the nature and version of the software or project.

1. **How long will it take dev to do X?** This is usually my first question, but it is loaded with caveats. The proverbial two line change by dev might take less than a minute to make, a few minutes to compile, and a few more to push into a branch for testing. But that little change in dev could have large testing implications. On the other hand, a new feature or slew of bug fixes by a competent developer might not require much qa time. So the point here is understand how applicable the answer to this question is to the testing tasks.
2. **How long did it take last time?** Has the feature or release or whatever existed in some form before? If you are working on something brand new, this question is not useful. History can be a useful guide to future test durations, yet only if the circumstances are somewhat similar. If a feature has not been changed, or changed very little, then it will need *less* testing time than

i. A sophisticated desktop publishing application first released in 1988. Frame was acquired by Adobe in 1995.

Copyright ©2010 Blake Elder

before. If development is making significant changes (I'll let you figure out how to quantify 'significant'), then you will need *more* time. But how much more?

3. **How long did it take the other team?** This is a variation on number two above. At Frame we supported three major platforms: UNIX, Mac, and Windows. Frame always released the UNIX[ii] versions first, so the Mac and Windows teams knew in detail how long dev and qa took for that first release. Now task duration, for say, testing footnotes, might not take exactly the same time as on UNIX, but at least you had a starting point.
4. **How many developers, how many qa?** Unmentioned so far are staff size and time units, i.e. man hours. So when getting numbers from developers or other test teams or past projects, it's important to make sure you're using units that make sense across projects and contexts. If three developers spend seven days on feature X, how much time will two qa engineers need to test feature X? Solve and show your work.
5. **Configuration testing requirements**. The unique burden of qa. Over the years and across companies this has meant different things, but in every case it's a lot of work.

- At Frame qa tested on SunOS, Solaris, HP-UX, Mac OS for CISC, Mac OS for RISC. Windows 3.1, maybe some others. In addition, qa needed to make sure FrameMaker worked well with other key publishing and printing software, such as Adobe Type Manager and various font packages.
- At Visioneer we built scanners and software. We needed to test on Macs and Windows systems, and also through a variety of hardware ports: COM port, parallel port, SCSI ports, etc. For fun you could daisy chain some hardware devices (Iomega was the most common) to make sure port contention worked fine.
- At Claria we supported only Windows, and tested Internet Explorer, Firefox, and OEM versions of IE, such as from AOL or Yahoo. In addition, qa kept busy with various connection types: dial up, ISDN, DSL and it's variants, cable modems, etc.
- At Playlist it was a combination of operating system (OS) and browser. Testing was prioritized based upon Google analytics reports of our several million per day visitors. At the time it was roughly WinXP, Vista, Mac OS, running Internet Explorer, Firefox, and Safari (Mac only).

You get the idea. Within each of these, there were always more questions: how different is Win98 from Win95? Or more recently, Window 7 from Vista? What about that new version of Firefox? What market changes require you add or drop a platform? For example, in the early days of the internet most users surfed the web with either Netscape or IE. Today Firefox replaced Netscape, and you can add to your configuration list Chrome, Safari, and if enough of your audience uses it, Opera.

Also, how much configuration testing development does prior to submitting to qa? Usually it's not much. This situation is a bit logistical and a bit cultural. On the logistics side, it's often hard for a developer to have all necessary configurations available to debug on. On the cultural side, developers have their preferences. At Playlist most of the developers were on Macs and were not inclined to look at IE. The Playlist web site looked awful on IE, especially IE6, which comprised about thirty percent of the audience, until users started to upgrade.

---

ii. SunOS, Solaris, and HP-UX

Copyright ©2010 Blake Elder

Configuration testing is a huge topic, so make sure you understand it's breadth and depth. Once in a while perhaps the gods will smile on you, and all your end users run the same type of system. O happy day.

6. **Third party code, third party interactions?** If all the code you test comes from developers at your company, you can skip this section. Otherwise, there are third party dependencies of various kinds:

- Many shrink wrap applications could export their files to some other format. Intuit's Quicken, Frame's FrameMaker, and Visioneer's Paperport could all export their native file formats to some other format, thereby making the files more portable. Often the code or libraries to do the export were written not by the developers at those companies, but by third parties.
- One of my favorite features ever was PaperPort's Simple Search. Eugene Veteska, the developer, wrote his own code, then integrated a third party OCR program and also dtSearch's index and search code. The feature worked great when released, but engineering had to work closely with the OCR vendor and dtSearch to get fixes and changes.
- At Playlist we built a music track and playlist purchase system. A key part of the flow was the payment provider, Vindicia. Keeping them in the loop and working with their development team added considerably time, and delays to the project.
- At Playlist we relied heavily on Open Source software, especially memcached. The developer who owned this area was not fluent in this technology, and we had to hire a consultant to help sort out the problems, adding costs of time and money. And we had to customize memcached for our particular requirements.

So why does this matter? I believe the more outside code there is, the riskier your schedule is. Technology providers (third parties) often have other, larger customers, so your fixes may be delayed. More recently, open source code works very well and is fantastic for in many ways, but it may take your developer(s) a while to fully understand it, especially if they are working with new packages.

But, again, what does this have to with creating a testing schedule? Often neither the project manager nor the dev manager will factor in potential delays or turn around time when getting bug fixes for outside code. So at a minimum you must state as an assumption an expected turn around time for fixes for third party code. Even better would be to assume some lag in fix time and add in time to your schedule.

7. **Are you familiar with the thing?** You'll know much more about a feature and the required test time if you are familiar with the feature/module/product/thing. Try to take the code for a test drive before scheduling.
8. **Concurrent projects.** Often your team may be working one or two other projects. Will these take away time for the project you are scheduling? Are developers working on other projects, causing them to delay implementation and bug fixes? If conflicted, which project is more important?
9. **What type of testing is needed?** Maybe you're building and testing a quick prototype to deploy to a test audience. Maybe it's mission critical software. Just doing load testing on some servers? Or is it a regression of an English version localized into FIGS (French, Italian, German, Spanish)?

Copyright ©2010 Blake Elder

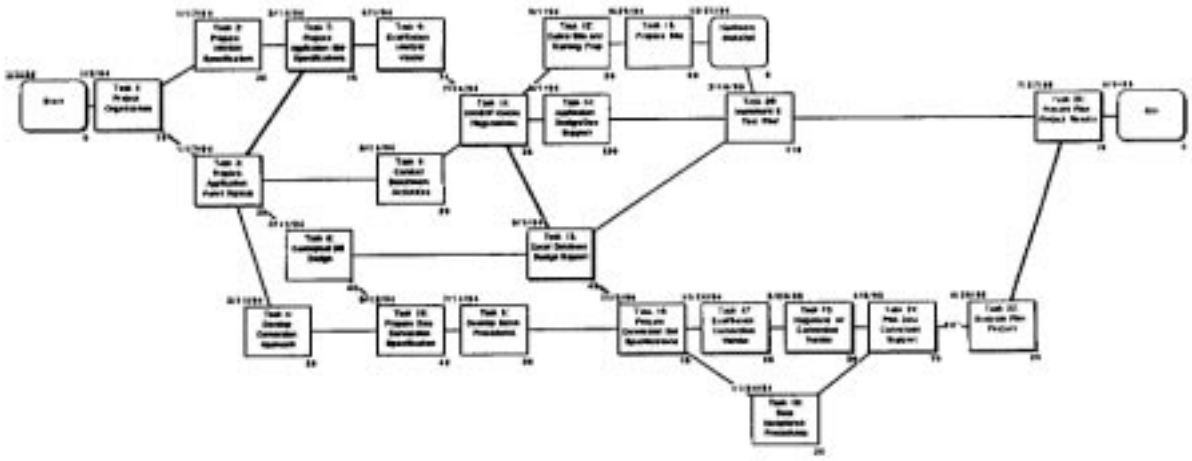
10. **The most important thing: the people doing the work**. Let's start with dev. You must know the ability of the developers who are writing the code you test and are fixing the bugs you log. As with so much, success depends on the quality of the developers. Did the developer provide good information about her feature? Does her code work when it comes into qa, or is there something wrong with it and it takes two or three tries? Are bug fixes delivered quickly, with good notes about testing? No amount of good testing will make up for a bad developer. Knowing the developers will help you make a more accurate schedule.
    The same applies to your test team.Understanding their strengths and weaknesses will help you create more realistic schedules.

The above list is not the last word in scheduling, but it's a start. There are books and websites that provide better parameters for creating a schedule. If looking around, start with something from O'Reilly. Of late I've liked Scott Berkun's *The Art of Project Management*.

There's one other question I ask, but it's usually directed at marketing, sales, or business development managers who are understandably eager to ship the next 'thing'. Of course, that question is **"When do you want it?"** Obviously the business considerations of a project are very important, but knowing a desired or expected release date is separate from figuring out how much test time you need. If the schedule you come up with is at great variance to the preferred business release date, then you must work with the team to reduce the scope of the release, revise the release date, or something else.[iii]

Tools

In the mid-1990's Frame managers used MacProject to create schedules. When I started making schedules, I used MacProject also. The most salient feature, and really the only one I recall, was the use of PERT charts for listing tasks, their owners, and dependencies. Below is an example of MacProject PERT charts, circa 1994.

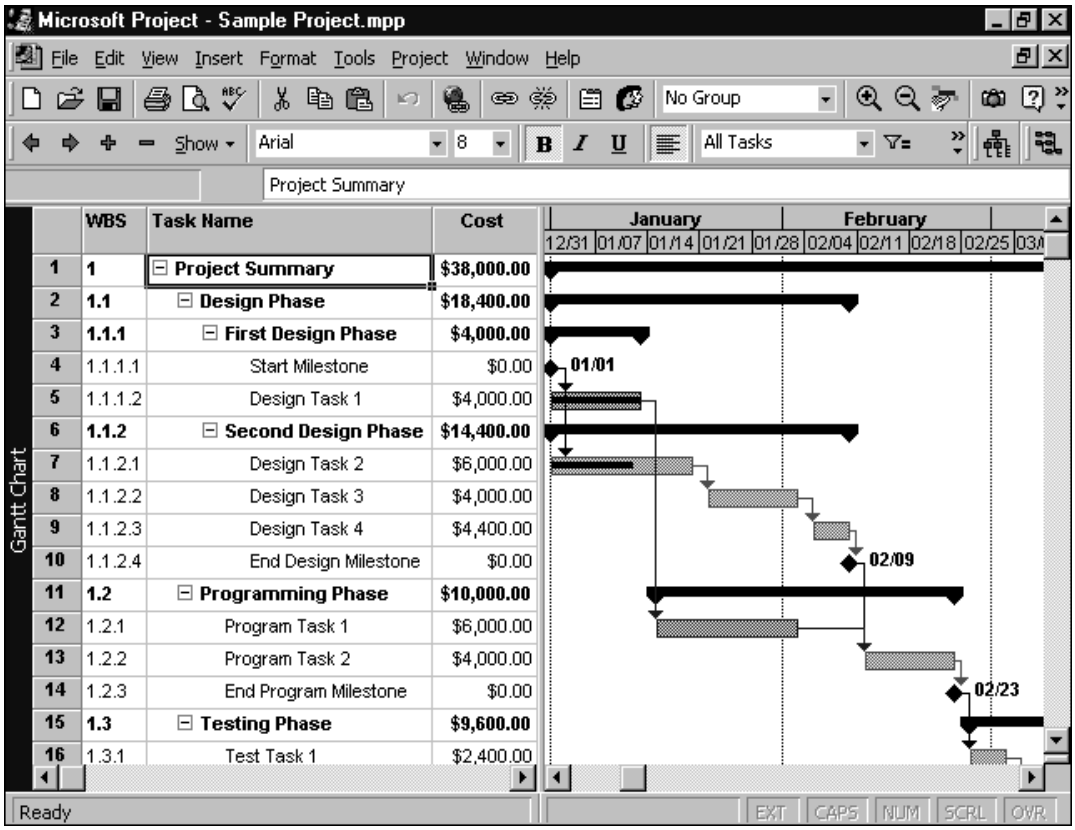


iii. A common problem across industries across the ages. One of the better software narratives on this topic is the classic *The Mythical Man Month*.

This view is kind of useful. And MacProject had a cool feature that would show (in red!) the critical path of a project. But generally, I did not, and do not find PERT charts very useful, especially for creating and modifying schedules. Most projects had many tasks, running into the low hundreds, and the MacProject interface and UI did not lend itself to easily managing large numbers of tasks.

One day walking by the office of a development director, Paul Bailey, I noticed he had a schedule on his wall done in Excel (easily recognized by the spreadsheet style). Column A was the task, column B the owner, Column C the duration. There might have been a few more columns. I liked the clean style of listing the tasks, and the easy way of typing in tasks and owners; MacProject was clunky. Paul said he preferred using Excel over MacProject for these reasons, and was also looking into Microsoft Project as an alternative scheduling tool.

I got a copy of Microsoft Project, for Windows, and started working with it. As much as one can like scheduling software, I liked MS Project. There was a spreadsheet style grid for entering tasks, making modifications was easy, and there were numerous views available: Gantt chart, Pert chart, etc. Now an experienced scheduling manager who had used other tools, such as Project Workbench, might have taken a dim view to my efforts and tools, but for me, it worked just fine. A view of MS Project:



Copyright ©2010 Blake Elder

Scheduling in action

About a year after becoming a manager at Frame, we went through a reorg, and then engineering was given a large project: a major release of FrameMaker on all supported platforms, with the same release date. Such a broad, simultaneous release had not been done before and required major coordination between the various platforms teams (dev/qa): UNIX, Mac, and Windows. The qa department had a new leader and she made me the schedule czar for all qa work. This was no great honor, but I learned a lot. The process was roughly:

- The qa managers and engineers created schedules by assigning qa people to tasks, setting the order of the tasks, and estimating the time needed to test. We went through this cycle a couple of times before we were satisfied.
- In MS Project I set up an engineering calendar to set the work day to six(6) hours, any only five days a week. In case this needs explanation: given typical interruption rates and the usual distractions such as meetings, an engineer working six solid hours per day on a tasks is a huge accomplishment.
- Once the project started, on Mondays I would give each qa engineer his schedule of tasks for that week. Call these “the planned” schedules.
- On Friday afternoons the qa engineers would send me the results of their work. Also known as the actual schedule. I would then update MS Project with the latest information, then re-level, and see how were doing relative to the plan.
- The qa schedule fed into the larger project schedule; dev and doc were the other groups qa marched with.

The Framemaker project took about a fourteen months, and the above described scheduling drill lasted for about eight months. The product shipped two months late.

Lessons learned

1. Under the best of circumstances, a well planned and regularly maintained schedule has about 85% accuracy with regard to how long tasks take, milestones (such as beta), and project completion dates.
2. Engineers are not very good at estimating task durations. Ditto for managers. Put another a way: accurately estimating durations in software is hard. Having previous schedules (planned and actuals) was very helpful for future planning, assuming reasonable similarity across the projects.
3. For a certain class of software projects, I am not sure it is worth the effort to create, maintain, and track a detailed schedule. There might be better indicators of the state of the software and readiness to ship/deploy, which is really the only milestone that counts. Moreover, for many web/agile style projects, the methods and tools described here are not appropriate.
4. Since leaving Frame, I have never created AND maintained a detailed schedule. I have used MS Project to create schedules at the beginning of a project. This is useful for creating detailed tasks lists, owners, and estimating durations. But once a project is started, I have not tracked actuals.
5. Creating these schedules can be a useful tool for managing up, especially executives. If nothing else, I have learned how to quickly create a detailed schedule and use it to whatever end I need, usually getting more time and people.

Copyright ©2010 Blake Elder

To agility and beyond

Ashton Tate, Intuit, Frame, and Visioneer published shrink wrap software. At these companies the schedules generally followed a waterfall method. Project durations were in the weeks and months, and could easily last over a year. Creating and maintaining schedules was generally helpful given the long nature of these projects.

Claria (aka Gator) was a bit of a hybrid between classic release methods and web/agile styles.All client side software distribution was over the internet. Claria's software, which interacted with the user's browser, ran locally on users's PCs. Because there was a client side installation, to certain extent we needed to be careful and thorough when building the software. But, and that's a big but, since delivery and updates were done over the internet, we had a bit more flexibility (agility?) in our releases. At Claria I occasionally created a schedule in MS Project, but this was done at the beginning of projects and never maintained.

After Claria I worked at Playlist.com, where the entire user experience was through the browser, a full web 2.0/agile environment. Our tools for tracking tasks were ScrumNinja and Fogbugz, but this was only for developer tasks. Given that I had a small team, and Playlist released weekly, I rarely created anything that could be called a schedule.

One of the primary scheduling advantages of the agile[iv] model at Playlist.com was the decomposing of larger tasks in to more manageable, trackable tasks. You would never see a developer working on something called "new website" for six months (I exaggerate to make a point). With an agile style project you know much sooner if something is falling behind, and therefore have a better chance to fixing the problem.

If you're a qa manager at a software company, chances are you'll be asked for a schedule in some form or another. Find a decent tool (Open Workbench looks pretty interesting) or web site (I've used Zoho recently and liked it) but don't spend endless hours on the drill. Understand the benefits, potholes, and reality of scheduling. The rest of your company needs to know what you are up to, and what you are doing, and when you are doing it. Provide them with this information in a clear, concise manner, but don't go overboard; you have too many other things to do.

Sources

Mac Project image from the article by Larry J. EngelKern, *Project Management Tips For AM/FM Project Planning and Implementation*.

Microsoft Project image from The Critical Tools web site (http://www.criticaltools.com/).

iv. To be sure agile, and extreme/xp and scrum organizations all suffer from their orthodoxies. My favorite is the requirement of creating a test before the code exists to prove the code does not exist. That's about as useful as counting the number of angels dancing on the head of a pin. I've also found agile to be a bit of an excuse to deliver sloppy code. And last, people really do spend entire meetings standing.

Copyright ©2010 Blake Elder