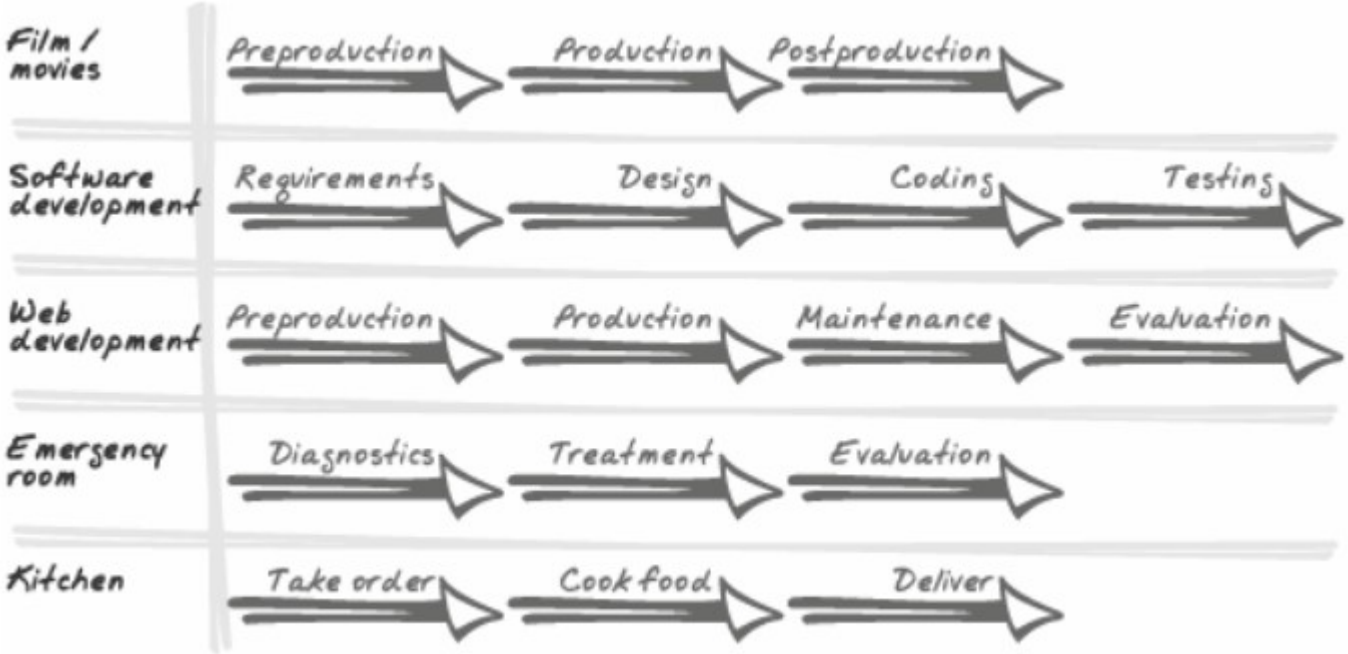
The Art of Project Management

In the abstract, many disciplines have similar processes. They all dedicate time to planning, executing, and refining. (However, you should never go to a kitchen for medical treatment or eat in an emergency room.)



«The Art of Project Management» By Scott Berkun

The balancing act of project management

- **Ego/no-ego.** Derive great personal satisfaction from work. But, project managers must avoid placing their own interests ahead of the project.
- **Autocrat/delegator.** In some situations, the most important things are a clear line of authority and a quick response time. A well-managed project should create an environment where work can be delegated and collaborated on effectively.
- **Tolerate ambiguity/pursue perfection.** The early phases of any project are highly open and fluid experiences where the unknown heavily outweighs the known. But at other moments, particularly in the later phases of a project, discipline and precision are paramount.

The balancing act of project management

- **Oral/written.** Despite how email centric most software development organizations are, oral skills are critically important to project management.
- **Acknowledge complexity/champion simplicity** The balancing act here is to recognize which view of the project is most useful for the problem or decision at hand, and to comfortably switch between them or keep them both in mind at the same time (without your head exploding).
- **Impatient/patient** Most of the time, the project manager is the person pushing for action, forcing others to keep work lean and focused. But in some situations, impatience works against the project.

The balancing act of project management

- **Courage/fear** One of the great misnomers of American culture is that the brave are people who feel no fear. This is a lie. The brave are those who feel fear but choose to take action anyway.
- **Believer/skeptic** There is nothing more powerful for team morale than a respected leader who believes in what she is doing. At the same time, there is a need for skepticism (not cynicism) about how things are going.

Confusing process with goals

Years ago, working on the Internet Explorer 4.0 project, I was the PM for several large areas of the user interface. I felt significant pressure: it was the largest assignment I'd ever had. In response, I developed the belief that if I could write everything down into checklists, I'd never fail. While things do need to be tracked carefully on a project, I'd taken it too far. I'd built an elaborate spreadsheet to show multiple data views, and the large whiteboards in my office were covered with tables and lists (and extra whiteboards were on the way).

Confusing process with goals

My boss let me run with it because things were going well. That is, until he saw me spending more time with my checklists and processes than I did with my teama big red flag (warning sign). He came into my office one day, and seeing the comically large matrix of checklists and tables I'd written on every flat surface in my office, sat me down and closed the door. He said, "Scott, this stuff is nice, but your project is your team. Manage the team, not the checklists. If the checklists help you manage the team, great. But the way you're going, soon you'll be using your team to help you manage your checklists."

So, instead of focusing on processes and methods, project managers should be focused on their teams

The right kind of involvement

Managers are not hired to contribute a linear amount of work to the factory or software shop, like a worker or programmer is expected to do. Instead, leaders and managers are hired to amplify the value of everyone around them.

Like a coach for a baseball team, the presence of a manager is supposed to contribute something different in nature from adding another individual contributor.

Take advantage of your perspective

For example, one morning during the IE 5.0 project, I dropped by Fred's office. He was arguing with Steve, another programmer, about how they were going to get the new List View control to work properlyan unforeseen compatibility issue had been discovered that morning. Neither one of them wanted to do the work. And from what I could hear, it would take a half-day or more to fix. I poked my nose in and confirmed what they were talking about. They nodded their heads, as if to say, "Yeah, why should you care?" I then told them to go talk to Bill down the hall. They again asked why, thinking this was a very specific architectural issue that I couldn't easily understand. I smiled and said, "Because I just left his office, and he has the new tree control working perfectly on his machine. He came across the problem last night and fixed it as part of another work item."

Summary

- Project management is everywhere and it's been around for a long time.
- If you keep a beginner's mind, you'll have more opportunities to learn.
- Project management can be a job, a role, or an activity (the advice in this book applies well no matter how you define it).
- Program management is Microsoft's strongly defined project management role. It is derived from the idea of a matrix organization.
- Leadership and management require an understanding of, and intuition for, several common paradoxes. These include ego/no-ego, autocracy/delegation, and courage/fear.
- Watch out for pretension and over-involvement in your management activity. The process should support the team, not the other way around.
- If you are a dedicated manager, look for ways to capitalize on your unique perspective of the team and project.

Schedules have three purposes

1. To make commitments about when things will be done

2. To encourage everyone who's contributing to a project to see her efforts as part of a whole, and invest in making her pieces work with the others

3. To give the team a tool to track progress and to break work into manageable chunks

A schedule cannot remedy bad design or engineering practices, nor can it protect a project from weak leadership, unclear goals, and poor communication.

Silver bullets and methodologies

Be very careful of how you apply whatever methodology you use: it shouldn't be something inflicted on the team. Instead, it should be something that supports, encourages, and assists the team in doing good work on the project.

Remember that the use of one methodology or another is never the sole reason for a project making or missing its dates.

What schedules look like. The rule of thirds

For any project, break the available time into three parts:

- one for design,
- one for implementation,
- and one for testing.

Depending on the methodologies you use, these phases will be called different things, or they may overlap with each other in certain ways, but all methodologies have time dedicated to these three activities. On any given day, you're either figuring out what should be done (designing), actually doing it (implementing production code), or verifying, analyzing, and refining what's been done (testing).

The rule of thirds. Piecemeal development (the anti-project project)

Simplest case possible: there is no project. All work is done on a piecemeal basisrequests come in, they are evaluated against other work, and then they are put into the next available slot on the schedule. Some development teams, web site developers, or IT programming departments work in much this way.

The rule of thirds still applies to these situations. Even if each programmer is working alone on small tasks, he is probably spending about one-third of his total time figuring out what needs to be done, one-third of his time doing it, and one-third making sure it works properly.

Divide and conquer (big schedules = many little schedules)

Where it gets complex is on larger or longer projects, where schedules are divided into smaller pieces, with each piece having its own design, implementation, and testing time. Extreme Programming (known as XP) calls these pieces iterations; the spiral model calls them phases; and some organizations call them milestones.

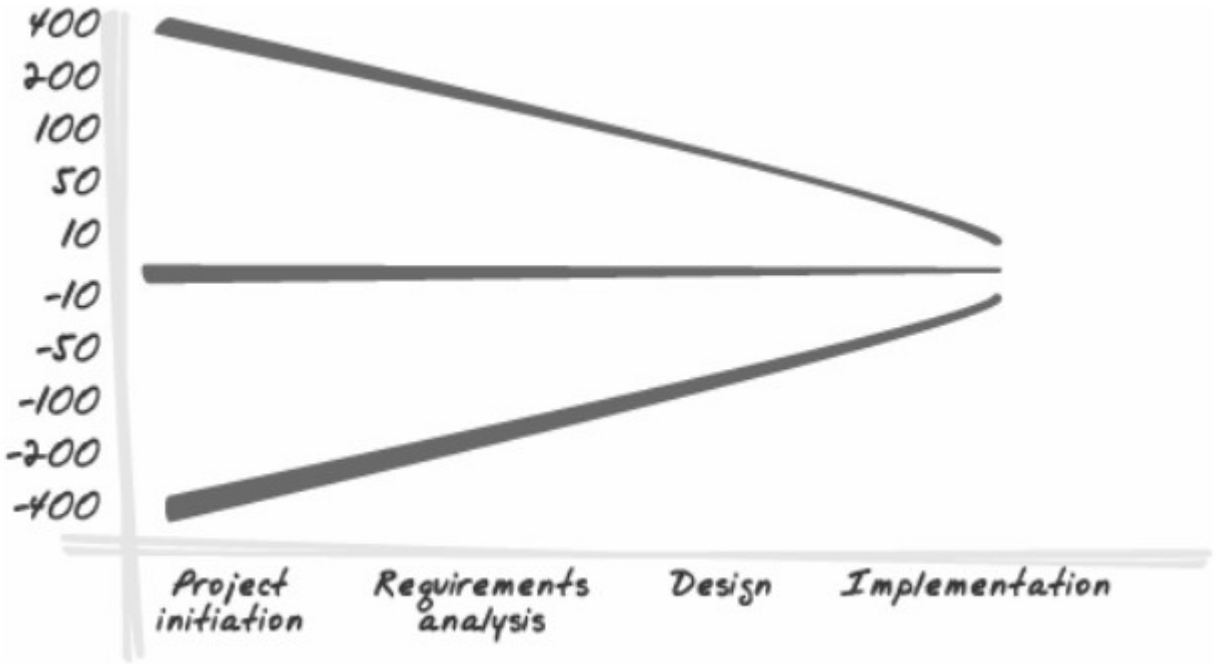
While XP implies that these chunks of time are only a few weeks, and the spiral model implies that they are months, the fundamental idea is the same: create detailed schedules for limited periods of time only.

Agile and traditional methods

XP and other agile methods assume the future is always volatile, so they bet on processes that incorporate direction changes easily.
Projects that have very high production costs (say, building a skyscraper, a video game console, or an embedded operating system) go the other way and invest heavily in planning and designing activities. It can be done, but everyone has to commit to the decisions made during planning, and the prohibitive cost for changes tends to be the only way that happens.

Most software development projects are somewhere in the middle

The range of estimation errors during projects (adapted from Boehm's Software Engineering Economics).



Barry Boehm, in his 1989 essay on software engineering, found that schedule errors scale in relation to how early in the project schedule estimation is done. If total schedule estimates are made early, they can be off by as much as 400%, in either direction (I suspect the errors are skewed against us, tending to take more time than we expect, although his data didn't show this).

The world is based on estimation

One thing that makes scheduling difficult is that few people enjoy estimating complex things that they will be held accountable for. It's always fun to brag and make bets about our skills ("This book/movie/web site stinks: I could make one soooo much better"), but when we're pressed to step up and deliversigning our names on a contract detailing our responsibilitythings change. We know that it's entirely possible that whatever we commit to doing today might be impossible or undesirable to do when that time comes.

Good estimates come from good designs

To the credit of programmers everywhere, the most important thing I've learned about good estimates is that they only come from credible designs and requirements. Good engineering estimates are possible only if you have two things: good information and good engineers. If the specs are crap, and a programmer is asked to conjure up a number based on an incomprehensible whiteboard scribbling, everyone should know exactly what they're getting: a fuzzy scribble of an estimate.

Here are some additional ways to ensure good estimates:

- **Establish baseline confidence intervals for estimates.** A guess = 40% confidence in accuracy. A good estimate = 70%. A detailed and thorough analysis = 90%.
- **Lead programmers must set the bar for quality estimations by asking good questions and taking wise approaches that the team can emulate.** Do whatever is necessary to kill the motivation for snide comments or backpedaling (e.g., "Don't hold me to this," "It's just a guess," etc.).

Here are some additional ways to ensure good estimates:

- **Programmers should be trusted** Sometimes, pressure has to be applied to keep people honestbut only as a balancing measure.
- **Estimates depend on the programmer's understanding of the project goals**
- **Estimates should be based on previous performance**
- **Specification or design quality should be to whatever point engineering needs to make good estimates**
- **There are known techniques for making better estimates**

List of questions that have helped to catch potential schedule problems early on:

- Were sick days and vacation time for all contributors included in some form in the schedule?
- Did individuals have access to the schedule, and were they asked to report regular progress (in a non-annoying way)?
- Was someone watching the overall schedule on a daily or weekly basis? Did this person have enough authority to ask good questions and make adjustments?
- Did the team feel ownership and commitment to the schedule? If not, why? Did the team contribute to the definition of the schedule and the work to be done, or was it handed down to them?

List of questions that have helped to catch potential schedule problems early on:

- Did team leaders add more feature requests than they helped eliminate? Did team leaders ever say no to new work and provide a reasonable philosophy to the team for how to respond to new (late) requests?
- Were people on the team encouraged to and supported in saying no to new work requests that didn't fit the goals and the vision?
- What probabilities were used in making estimates? 90%? 70%? 50%? Was this expressed in the master high-level schedule? Was the client/VP/customer aware of this?
- Was there discussion of another proposal that took more time but came with a higher probability?

List of questions that have helped to catch potential schedule problems early on:

- Were there periodic moments in the schedule when schedule adjustments and renegotiations could take place by leaders and management?
- Did the schedule assume fewer working hours over holiday seasons? (In the U.S., Thanksgiving to Christmas is often a low productivity time.) Are any highly probable disruptive weather events weighed into the schedule (for example, blizzards in Chicago, tornados in Kansas, sun in Seattle)?
- Were the specifications or design plans good enough for engineering to make good work estimates?
- Was engineering trained or experienced in making good work estimates?

The snowball effect

What must happen for schedules to work

- **Milestone length should match project volatility.** The more change that is expected, the shorter the milestones should be.
- **Be optimistic in the vision and skeptical in the schedule.**
- **Bet on design.** The process of design is the best insurance against ignorance and unexpected challenges.
- **Plan checkpoints for add/cut discussions.** Schedules should include short periods of review where leaders can review current progress and account for new information or customer feedback.

What must happen for schedules to work

- **Inform the team about planning philosophy.** Whatever schedule approach or technique is used, it should be common knowledge to the team.
- **Gauge the team's experience with the problem space.** One of the magic variables in scheduling is how experienced the team is with the kind of problems it is being asked to solve.
- **Gauge the team's confidence and experience in working together.**
- **Take on risks early.** If you know that Sally has the most complex component, deal with those challenges up front in the schedule.

Summary

- Schedules serve three functions: allowing for commitments to be made, encouraging everyone to see her work as a contribution to a whole, and enabling the tracking of progress. Even when schedules slip, they still have value.
- Big schedules should be divided into small schedules to minimize risks and increase the frequency of adjustments.
- All estimates are probabilities. Because schedules are a collection of estimates, they are also probabilities. This works against schedule accuracy because probabilities accumulate (80% x 80% = 64%).
- The earlier that estimates are made, the less accurate they are. However, rough estimates are the only way to provide a starting point for better ones.
- Schedules should be made with skepticism, not optimism. Invest in design to shed light on assumptions and generate reliable confidence.