The Art of Project Management

You won't be able to do all the tasks in a development yourself. Usually, you will have a team working with you, perhaps an external supplier, and you will also have to get what you need from the users and your managers. To fulfil your task you will have to manage, either directly or indirectly, the efforts of all these people. Essentially you must:

- *Lead!*
- *Plan!*
- *Communicate!*
- *Build the team!*
- *Get things done!*

This chapter is about how to manage people to achieve your project's objectives.

I assume that, as a newcomer to project management, you're going to be leading a relatively small project, or a team within a larger project. The principles are the same for larger projects, but there are some detailed techniques you'll need to learn, and you'll need to gain practical experience with a smaller job first.

What Does a Project Manager Do?

I've met many people who seem to think that project management is about solving problems of someone else's creation, like Tommy Lee Jones's character. However, I'd say that if this is your situation, then you're either a top trouble-shooter, or you haven't managed the projects properly.

> "I got tired of coming up with last minute, desperate solutions to impossible problems created by other 💣☠✌✸ing people!"
> Tommy Lee Jones in "Under Siege"

To avoid ending up like this, there are a number of things you have to do, regularly and effectively, starting with your first involvement in the project:

➢*Leading and building the team*. Co-ordinate the activities of a group of people, so that things happen in the right way and at the right time. Perhaps even more importantly, you need to co-ordinate their different *goals*. If they have a common objective, they will deploy their own efforts more effectively. Your aim is to build a team whose ability as a whole is much greater than the sum of its parts.

➢*Planning*. Every task on the project, and the use of every resource (including people), should be planned in advance. Otherwise, you will either under-use your valuable resources, or you will have insufficient resources and the project will slip. As well as managing time and effort, you have to manage the quality of the work which is done, and allow for things which could go wrong. You will have to regularly update your estimates and plans as circumstances change.

➢*Communicating*. Everyone must understand what is being done, by whom and why, and this is down to communication skills. You need to open up as many communication channels as possible, and make regular use of them.

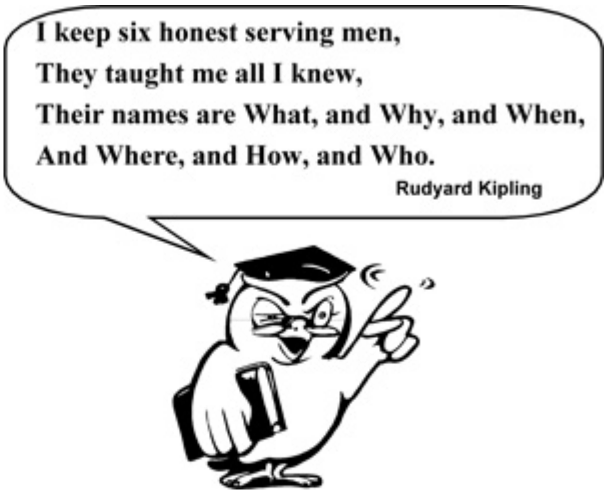
➢*Monitoring and reporting progress*. Constantly monitor progress (against your plan), spot problems quickly, and report on your progress to your users and managers. If you do this, and particularly if you can suggest practical solutions to the problems, then others will have greater confidence in you, and you will get approval for your work more easily.

➢*Get things done!*. Understand your overall objectives (and the intermediate deliverables), and progress towards them.

What are the Key Skills of a Project Manager?

Planning and *communicating* are the two key skills essential for success in project management. Like any other skill they can be learned, and will improve with practice. Don't be afraid to admit that you may need help with them. Watch those who do them well, and ask for advice. You can also learn a lot from books or training courses, but these are no substitute for experience!

Make sure that every part of the project is represented in a documented plan. Remember, a plan is only complete if you can answer the six key questions *what?*, *why?*, *when?*, *where?*, *how?* and *who?*.


I keep six honest serving men,
They taught me all I knew,
Their names are What, and Why, and When,
And Where, and How, and Who.
Rudyard Kipling

If *you* can answer these questions, check that your team, your manager and your users can answer them too. Make your plan one of your ways of communicating with people — it's not just something produced to satisfy moaning management with nothing better to do than to chase you for useless deliverables!

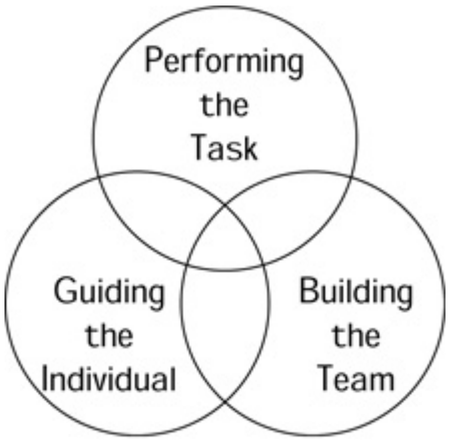
Keep asking people what they are doing, what progress they are making (against the relevant plan), and tell them what you (and the rest of the team) are doing. There are various ways this can be done: formal, written reports are essential, but verbal reports and memos are also useful. Don't, for example, wait until monthly report time to tell your managers about a developing problem which may need their help to sort out!

Note that I haven't said anything about analysing, designing or programming. These are not project management jobs. *You* need to make sure that they happen, and are done properly, but it is not essential for you to do everything. Indeed, you must avoid the situation where the real project management jobs are left undone because you're concentrating too much on the technical problems. Remember, you don't have to be a technical expert to manage a project. It's sometimes said that "management is the art of delegation" — more about this later.

In a small team, you will have some technical tasks (and you may be more experienced technically than the people you're managing). That's fine, but then you must plan your time carefully: some will be dedicated to your technical tasks, but some (quite separate, and planned) *must* be *dedicated* to managing the project.

How Do I Lead?

First, make sure that the overall objectives of the project are clear, and communicate these to the team: *focus on a common goal*. Develop and communicate plans which detail the tasks and individual goals of the team (and those around them). Finally, make sure that the work is being done and goals met, and communicate this progress against the plan back to IS management and the users.



Leadership

Leadership is often portrayed as a three-part process: performing the task, guiding and building the skills of the individuals in the team, and building the team as a whole. The results of the team should be greater than the sum of the results from its parts.

How Do I Get People to Do Things?

You should have a certain amount of authority over people working within your team because of your position, but this doesn't necessarily mean they will follow your instructions. Other people, outside your project team, will also have to contribute to the success of the project, but you have no authority over them at all. Thus you can't rely on orders. Instead, you have to persuade people to fulfil their tasks willingly.

Try to understand what motivates people in general, and each member of your team in particular. The most powerful motivators for most people are recognition, achievement and appreciation — not physical rewards like money (as long as the latter are adequate). This is good news for you: unless you're quite senior you can't do much about the salaries of your team, but you can make clear, formal and public recognition of their efforts and achievements. You'll find that even a mention by name in your progress report will get you some good will, and with a bit of ingenuity you can do much more than that.

Study what can de-motivate people. Don’t assume that completing the task or improving the company’s business are the prime objectives of all your team members — they aren’t. You need to understand what drives each person, and what they want out of life as a whole. For example, you may have two very good workers, one who has a busy (but cheap) social life, and one who has an expensive but solitary hobby. The latter will happily do lots of overtime, but the former will probably be very unwilling if you don’t take his other arrangements into account.

Find ways of aligning individual objectives with the team’s. Seek out what the objectives of each person are, then clearly communicate the team’s objectives and explain how meeting those objectives will result in progress towards the individual ones. For example, successful work on this project might put them first in line for work on a much more interesting project starting in a few months’ time — this sort of promise (if honoured) is a very strong motivator.

You won’t always be able to align all the objectives, particularly if the project is under real pressure to deliver. If you can’t, then the project’s objectives should come first. However, following the same principles may make a bitter pill a bit easier to swallow.

Beware of conflicting pressures. A programmer who should do something to improve quality but is late has two opposing negative pressures: criticism for either poor quality or late delivery. He will take the action which is “least negative”, probably the poor quality route. If this isn’t what you want, then, in the words of the song, you have to “Accentuate the positive, eliminate the negative” and create an environment in which a balance of motivations achieves what you want.

How Do I Allocate Work?

Divide the work up into separate tasks, each with a clear objective. You should then allocate these to people who have some reasonable chance of success. Try to give people a fair mix of the boring and interesting jobs, and allow them to grow by giving them a bit of a challenge (but not one they have no chance of meeting). Try as well to balance the work-load — don’t overload your best people leaving the less able with nothing to do. (See the section on Planning and Estimating for more details.)

Don’t be afraid to delegate. Sharing your tasks with other members of the team helps them to learn and understand what is going on, and may be essential if your own workload is too large. However, delegation does not mean that you abandon responsibility for the task. You must establish that the plans are being met and the quality is adequate — you have to *plan*, and spend time to monitor the work and possibly educate your team.

Make sure you can assess the quality and completeness of every deliverable. If you can't do it personally, get another member of the team, or even an outsider, to do so. If you have an independent QA group, or there's another project team working on a similar project, ask them to participate in the reviews and inspections.

If you follow these rules, and create the right motivation by matching the individual and team objectives, there's a good chance that your team will accept their tasks by consensus, and you won't need to give *orders* at all.

What if People Make Mistakes?

Recognise that everyone's human, and errors do occur. Don't be too defensive. However, don't create an environment in which sloppy work is acceptable. Instead try to create an atmosphere in which errors can be freely discussed, and where the identification of a problem creates a joint responsibility for the "finder" and "owner" o correct the problem and to prevent its recurrence. This must be even-handed — don't be afraid to submit your work for review by members of your team, or others.

Don't try to correct things that you might be slightly unhappy with, unless they are actually errors. The fact that you can think of a better way of doing something is not necessarily a reason for change. Remember that all changes cost time and money, and may impact the team's ability to meet its objectives. It's also very difficult for a more junior team member to make progress if his work is constantly being replaced by his seniors' "better ideas".

Be aware of the possible impact of poor design decisions or inadequate analysis on the project. Some such errors (which have probably been made by more senior members of the team) could prove fatal if not corrected early enough. The question to ask in these cases is not "can it be done better?" but "what are the risks if we adopt this approach?". In the case of poor analysis or design the risks will be unacceptably high, and you must manage them by working together to find a better way. Similarly, code or documentation which will cause problems for other people needs to be rectified. You must allow for some individuality — people aren't machines — but this must not be at the cost of maintainability or control.

On a practical note, most people will do a job in the easiest way. If you want to work to standards, then a small investment in simple tools and templates (for example to create a standard screen form, or for standard documents) will dramatically improve the chance of their acceptance.

The creation (and promotion) of the right attitude to errors is essential. Some errors will occur, and are probably essential if the team and its members are to learn. However, they should be identified by the review and testing processes, and corrected in a precise and non-personal way. Nothing is more demotivating for the team than to think that any old rubbish is acceptable. Allow the product to rise to a level of quality which satisfies the team — this *will* be productive.

Ultimately, you have to trust your people. Don't try to over-prescribe their actions. It won't work, and you will lose more in lost enthusiasm and motivation than you will ever regain in control.

How Do I Build a Team?

If you're lucky, then the members of your project team will start to work together in a way that will make the team more successful than the individuals working together. There will be a clear common objective, and work will be directed effectively towards that objective. You may find there is a certain amount of unspoken communication, with a greater common understanding than usual. This is a very enjoyable and productive experience. It is often described as a team *jelling*.

There's no simple, foolproof way to make a team jell, but you can set up the right environment, take sensible steps towards building the team, and hope for the best.

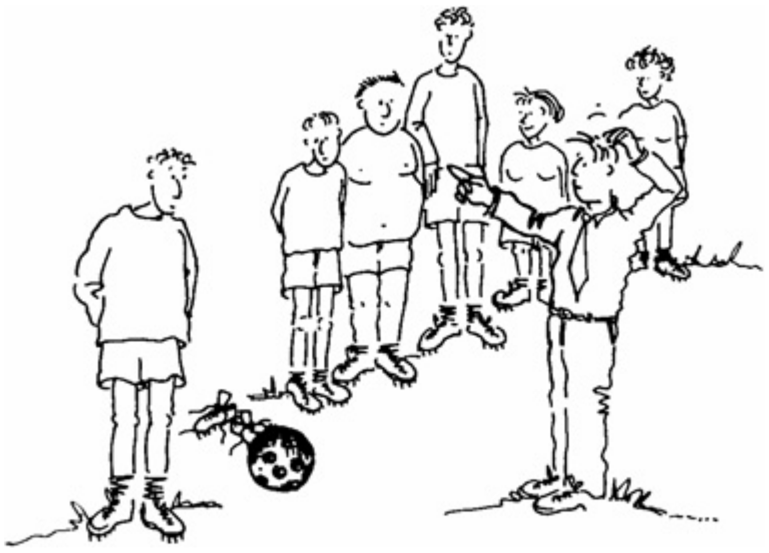
Team building starts with having an open framework in which everyone understands what he is doing and how it fits into the whole, and in which you can publicly discuss progress and problems. Try to build an identity for your team — your planning and reporting should be on behalf of the team, not "your" project. Encourage your team to behave as a unit: going for a drink or a meal together can often help build team spirit, and open up the channels of communication.

There are certain things you need to avoid if you're going to have any chance of making a team work effectively together. Defensive management and a bad attitude to errors and the quality of the product is one: see the previous page. Excessive bureaucracy and paper-heavy development is another: you should try to get the whole team to understand the principles of targeting each deliverable and delivering just what the target audience needs.

It's almost impossible to build teams if the members are physically separated, or if each member's time is fragmented between a number of jobs. It's worth fighting battles with your management for a good working environment in which the whole team can work properly together. A lot of the research suggests that the working environment is the only factor which consistently affects quality and productivity. A good working environment has to include *quiet* space for the high-concentration activities such as programming or writing, and separate space for people to work together without disturbing the rest.

📖You can't *make* a team jell, but you can provide the right environment. This section (and the next one) are an introduction to the most important principles. Read *Peopleware* if you really want to manage people and teams properly. The working environment is one of the most important factors, and may not be under your control. If your managers won't do something about a poor environment, try to get them to read *Peopleware* too!

How Do I Make Sure the Team is Complete?



In order to achieve its main objectives, your team has to do a lot of separate things. These tasks won't all appear on your official list of tasks in your plan, but they need to happen nonetheless if your team is going to succeed:

- *Leadership*. Someone (probably you) needs to align the goals of the team members with the team's own objectives, and provide direction towards those objectives. On a larger project, you'll need other leaders for each separate part.
- *Decision making*. Someone needs to make decisions when necessary. This needn't be the same person all the time, and needn't always be a single person. In some of the best teams, there is some form of collective decision making, but the leader has to be able to make and follow a decision if the collective process doesn't work for any reason.
- *Team building*. This is one task you can't make explicit (" Alice, you're in charge of making sure the team jells this week!"), but you will often find that someone takes charge of arranging little social events, or smoothing over ruffled feathers! If you have such a person, who may just act as a catalyst for the other members of the team to sort things out, then it's invaluable.
- *Communication*. As we've discussed, this is key to a successful development.
- *Selling the products of the team*. I don't mean the external sales of what you have produced, but the effort of proving to management and users that what you have produced is what they want and asked for. This is closely allied to communication, and may be done by the same person, as part of the same tasks.

- *Ensuring technical coherence*. Someone needs to own the high-level design (or architecture), check it is being followed and check that the elements fit together into a coherent whole. From the conception of the solution and the basic design through to integration with other systems, someone has to understand the structure and communicate it to other people. This role of software architect is so important I devote a whole section to it later in the book.
- *Planning*. Someone needs to create and maintain the plans and estimates.
- *Productivity and progress checking*. You need to make regular checks on the progress of each activity against the plan(s). If you are trying to *measure* your achievements, someone needs to be in charge of the measurements. Someone needs to make sure that you meet your targets, and jobs are actually finished — the formal announcement of targets met can have a very positive effect.
- *Quality checking*. Each component or deliverable has to be checked against its specification. Someone other than the object's creator should do this checking, and you will find that some people have much better checking skills than others, just as some are better programmers or writers.
- *Documentation*. Even if you're careful with the paperwork load, there's a lot of documentation to be created and maintained.
- *Administrative & secretarial functions.*

These tasks require a range of personality traits and skills that you won't find in any one person, so you need to build a similar variety into your team. Take advantage of the different strengths within people. If you are aware that your own communication skills are not as good as those of another team member, involve him or her in helping you to write and present reports.

Be sensible with the documentation and administrative functions. If you're lucky enough to have a professional administrator, then use him or her. If not, share the admin. tasks, but make sure they get done. Whatever you do, don't tie up your best technical resources in paperwork, just because they've risen to the "senior" position of team leader.

Leadership and management skills can be learned, but don't exist just because of seniority or experience. Don't promote a good engineer to be a poor team leader.

Together with your managers, you are responsible for staffing your team. Look at the list of tasks above, and make sure that someone in the team is tackling each one. If not, you have a profile for the missing members of your team.

How Do I Plan, Report and Communicate?

How do I know if I'm communicating properly?

Talk to people. Spend some time just talking through the job with each member of your team, your managers and your users.

Concentrate on making communication effective: see if your team, your managers and your users can answer questions *you* ask them about *your* plans. If they can, then you are succeeding. If not, then you may need to try something different.

Be aware that typically you only remember 20% of what you hear, 30% of what you read, but about 50% of what you both hear and read. Thus you should try to have both written and oral communication together: *always* write a confirmation of a discussion, telephone call or meeting, and whenever practical follow up a written communication with a verbal clarification. This is particularly important if your team has a mixture of first languages, or widely differing communication skills. It's also useful in resolving disputes, and essential when working with external suppliers.

In written material, make sure that the structure is clear so that people can refer to it more than once — they may need to try to find the 50% they haven't remembered! When you're dealing with your users this is very important — the style of communication may be different but the rules should be exactly the same.

Make the most of meetings. Make sure every meeting has a clear agenda and objectives. Use them to build joint understanding of the problems and objectives, and to reach a consensus about the way forward. *Do* allow constructive discussion of relevant topics, but *don't* allow your precious time to be chewed up by arguments or technical discussions between two or three people — the simple solution is just to suggest that this is a topic for another meeting. At the end of the meeting, sum up what you believe has been agreed. Make sure that every meeting has minutes — if one of your team does the minutes you can also check (when you review them) that he has understood the main messages of the meeting. An *absolute minimum* is to document the action points and anything you have agreed on!

There's no set list of meetings you must have (although one may be dictated by your management or a contract). However, you should aim to have at least a weekly progress meeting with your team, which should have *documented* actions and policy decisions. You should also meet formally with your managers and (maybe separately) the users every month, and compare progress with your plan. Again, you should produce minutes stating what you have agreed, and who has what action points.

How Do I Plan?

Creating a plan is a separate topic (to which another chapter is dedicated). The trick is that you must try and identify every task required to complete the project. Then, for every task, you must be able to answer the six key questions *What?, Why?, When?, Where? How?* and *Who?*. If you can't answer all these questions by reading the plan, then it isn't complete.

In an effective plan the individual tasks must be broken up enough that they are fairly short (i.e. a few days), and there must be a clear yes/no or numeric measure of success. Then you can either be sure that a task is actually complete, or you can trap slippage early on.

How Do I Know What Progress I'm Making?

Once you have a plan, you need to report your progress against it. The most important thing here is to be absolutely truthful: be proud of your successes, but honest about your failures and any problems you may be having. If others know, they may be able to help, but if you hide the true state of affairs then by the time the problems come to light it may be too late to do anything about them! Know when to get help — there is nothing unusual, and no shame, in having problems and needing assistance, but you will fail as a project manager if you don't accept this.

Talk to your team and find out how they are progressing, *against the plan*. *Use* the plan as an active tool, not as a dead report which you have to update for your managers' benefit. Get regular status reports from every member of the team (listing their progress, plan and problems). These should only take them about 20 minutes to write, or they don't understand their plan. You can then use these in your own report.

The first 90% of the development takes 90% of the time. The remaining 10% takes the other 90% of the time

Be sceptical. For work in progress, ask "How long to complete", not "How much have you done". Don't fall into the trap of the old saying… For a job which is supposed to be finished, check that *all* the finishing off has been done (including any paperwork or delivery procedures) and you can *prove* the targets for that component have been reached. If you have an independent testing or QA team, then look for verification of progress from them.

Make sure that the users, your manager and your team are aware of any changes to the objectives or the deadlines. There is nothing worse than to waste time because someone has not understood that the priorities have changed.

How Do I Gain Management Approval and Confidence?

Earlier in the chapter, I discussed how to create an environment in which you can trust your team to work effectively, with some tolerance of problems providing they are properly handled. Create a similar relationship with your managers, as otherwise your own efforts will be hampered. The following suggestions will help you to create a reasonable relationship, although they cannot be guaranteed since so much is down to personal preferences and styles.

A stream of deliverables on time and to the requirements will produce confidence in you. This is one good reason for creating a number of interim deliverables, and delivering the system in phases, with simple, low-risk items being delivered early. A method for structuring the project this way is in the chapter “Structuring the Development”.

Unfortunately, you won’t always be able to achieve this. The project structure may not suit this approach, or you may run into problems. The key to creating management confidence is good reporting and a clear information flow. There are no prizes for hiding problems, but if you spot and report them early, together with a possible solution, then your managers should be happy that you are in control.

How Do I Introduce Proper Methods?

You may have a more difficult problem: you want to use proper methods and run the project as this book tells you to, but your managers may not support this. The following arguments are often offered against use of proper methods:

- *The method is too theoretical, and won’t work in the “real world”*. The answer to this is that unstructured “real world” techniques fail too often.
- *Everything has to be written down — there’s too much documentation*. There can be a grain of truth in this, as I’ve said earlier. However, good written communication is *essential*. A sensible level of paperwork will often save a lot of wasted time. The trick is to find a balance, by properly targeting deliverables and written communications.
- *We have always managed without lengthy procedures before*. If this is true, and all the earlier developments have been successful, fine. In truth, there is probably plenty of evidence that the informal developments have had their fair share of problems. If you can, investigate the true situation, but avoid creating ill feeling if your researches show too many past failures!

- *These are good methods, but we can't afford to implement them now*. Why not? If the informal methods are causing failed projects, and therefore wasting time and money, you may not be able to afford *not* to implement them. If it's a question of developer training and learning curves, then you need to concentrate on a few relatively small, high-benefit changes.

A lot depends on the maturity of your software organisation. Surveys have shown that about 90% of software organisations are in a state best described as *initial* or *chaotic*. There is little formalisation of the development process, with only ad-hoc controls and irregular use of tools and standards. Some developments do succeed, but success is neither repeatable nor measured. Other organisations do have a more formal development process and some technical standards, but very few (if any) have reached the optimised, measured quality regime of the best in other industries.

The only way to move up from the chaotic level is to introduce good basic project and quality management practices. Advanced technology (such as better CASE tools, or complicated methods) won' t help much, and even if it does solve some problems the improvement won't be measurable, repeatable or general across the organisation. In the worst case, attempting to make big changes to an immature organisation by using a particular new tool or technique could cause drastic problems. If you're in that position then all you can do is concentrate on good basic project management.

On the other hand, if you're lucky enough to work in an organisation with defined management standards, then work within the existing framework. Concentrate on changes to things like methods and technical standards, fitting them into the existing practices.

You have to make your changes fit the corporate culture. Changes are easier to adopt if they can be understood in the context of what already exists. For example, if your organisation makes good use of electronic mail, don't try to introduce a method based on paper forms. The corporate culture will also affect activities like team building and your leadership style. If possible, try to work within the spirit of the organisation and make a series of small changes, rather than large changes which are likely to be unacceptable. If there are a lot of other changes happening in the organisation, then it's probably not a good time to make too many changes of your own.

Explain to the managers *why* some things must change. Try to fit the changes to the current corporate culture. They should then support you. If, however, there is no prospect of long-term improvements then you may be better off elsewhere.

📖*The Decline and Fall of the American Programmer* discusses at length the SEI software process maturity model, and what you have to do to move up it.

How Expert Do I Have to Be?

You don’t have to be an expert in technical matters to be a good software project manager. As we’ve already seen, most of the problems are related to the management of people, and you won’t solve these by technical solutions. Instead, you have to concentrate on managing the people: by understanding their needs and objectives, by creating and maintaining the right management framework, and by leading, planning and communicating.

You don’t even have to be an expert in management techniques. Most of the approaches in this book require some common sense and sensitivity, but the detailed methods are quite straightforward and don’t need any great tool support or prior knowledge. Obviously your ability will grow with experience, you need to constantly evaluate and improve your skills in communication, estimating, planning and man-management, and you may need to get help from time to time, but you can start from quite a low baseline.

However, this doesn’t mean that you can succeed without any technical knowledge at all. There are a number of things you must be able to do which need a combination of technical and management skills:

- *You have to be able to evaluate the accuracy of information given to you*. You may have a team member who says “I can’t do it because…”, or you may have to arbitrate between two conflicting technical views. In these and many other cases, you’ll either need to understand the technicalities, or find a way of reducing the information to a form you can understand.
- *You must understand what’s going on*. You have to report on behalf of the team to users and managers. You must therefore be able to understand the state of the project. In some cases, you will also have to translate quite technical reports so that less technical people will understand them.
- *You may have to provide guidance*. It’s unlikely that a developer will expect you to provide a detailed solution (if you don’t have the technical expertise), but you will sometimes be asked for guidance.
- *You must be sure that instructions are practical and estimates realistic*.
- *You must understand how good progress is*. If you want to avoid the “90% complete” syndrome, you must understand what has actually been achieved, and, more importantly, what you still have to do.

The following strategy may help you deal with those problems where you don’t understand the technical detail:

1. Make sure you understand the software production process. A lot of questions will be answered by the team themselves if they follow the methods properly. In other cases, the problem isn’t technical, but will arise because the job is being done out of sequence, or without the proper background information.
2. Try to get independent advice. If you have access to an independent reviewer (for example in the QA group) then use him or her to evaluate information and explain things to you. If not, canvas opinion from other members of the team. It’s important to show that this is not just checking for errors, but an important part of making sure there is a joint understanding of the situation.
3. Get the person(s) asking the question or presenting the problem to find and assess a number of options for its solution. Use risk analysis techniques (see next section) to evaluate the impact of each option on the project, then choose the option which has the greatest benefit/risk ratio.
4. Try to reduce what you are told to a form you do understand. You may be able to work by analogy, or get the members of your team to use simpler examples.
5. Ask the right questions. If you don’t understand the answers, you may need to ask different questions.

The last point is probably the key. If you ask the right questions, then the answers may be obvious. My father relates the following conversation between a very bright, but blind, director of Imperial Tobacco and a telephone sales girl in a new company which had just been set up and which was to be very successful:

“Good morning my dear. You telephone the customers to get their orders?”

“Roughly how long does each call take?”

“I see, around three to five minutes.”

“And you work from nine until five?”

“Can you look up on yesterday’s log to see how many calls you made?”

“Twenty-five. Thank you my dear. Good morning.”

Keep probing with different questions, and the answers will add up and make sense!

How Do I Spot Problems?

You have to go looking for them! If all problems announced themselves in advance with a big fanfare, then you'd just have to find a solution to each one and put it into practice at the right time. This would be a lot easier than what happens in reality, where problems tend to creep up while you're not looking, and you don't notice them until too late.

"If anything can
go wrong, it will"
Murphy's Law

As a project manager, you have to be a bit pessimistic. You can't assume that everything will always go right — it won't! Murphy's Law is probably a bit too pessimistic, but it's not a bad basis on which to manage.

Remember as well that it's human nature for most people to hide problems when they first see them, thinking either "maybe it'll go away", or "maybe I can think of a solution to this and recover without anyone knowing".

You can't manage like this. You have to try to spot problems in advance, and make plans which will cater for them with the minimum of disruption to the project. You have to encourage your team to tell you, early, about any problems they can foresee, and in turn you should notify your managers. You won' t be criticised for thinking about problems (particularly if you're also thinking about solutions at the same time), and you may even be praised for it. However, if you hide problems until they are having a serious effect, your managers and users will rightly accuse you of bad project management.

You have a couple of weapons in your armoury. The main ones we have already discussed: they are communication (with everybody, and frequently), and the use of formal reporting & interim deliverables to provide you with clear information on progress (or the lack of it).

The other thing you can do is perform a formal *Risk Analysis*: You need to ask yourself the following questions:

- What are the potential problems?
- What is the probable impact of each problem?
- How likely is it to occur?
- How can I prevent it or identify it early?
- What do I do if it does occur?

The best way to do this is to have a brainstorming session with the team (and/or your managers and users) to get a list of possible problems. Assess the impact and likelihood of each on a scale of 1 (low) to 10 (high), and multiply the two numbers to give you a severity rating. Then, for the most severe potential problems, develop an action plan (for prevention if possible) and an outline contingency plan (in case you can't prevent it). Give someone on the team the job of tracking any indicators you can think of, so you get the maximum warning of the need to activate the contingency plan.

The following spreadsheet shows part of such a risk analysis table:

Potential Problem	**Impact**	**Likelihood**	**Severity**	**Tracker**	**Action/Contingency Plan**
Insufficient disk space	3	3	9	Fred	(A) Investigate cost & lead time for upgrade
Insufficient memory	2	2	4		(No action at present)
Version 1.4 of database is too buggy	9	3	27	Joe	(A) Send Joe on early training course. (C) Use V1. 3
User requirements not completed on time	8	5	40	Fred	(A) Check progress and choose best areas for

					phased work
Users not available for testing	6	6	36	Alice	(A) Check user plans (C) Use external testers?
Late delivery of external subsystem	9	5	45	Alice	(C) Will need to build simulator for testing
Response time inadequate	7	6	42	Joe	(A) Prototype key transactions. (C) Consider second processor

Remember this is a plan, and needs to be updated and reported against like all your other plans. The “trackers” should report the state of the indicators to you as part of their progress reports. As dangers pass, you can scale down the severity of a problem and maybe drop it off the list, but you also need to review the list regularly and see if anything new needs to be added.

Is Programming Important?

The majority of software developers start their lives in the industry as programmers. You may well be one such person. Thus it's very easy to consider programming to be the core activity of software development. It takes quite a lot of effort and self-discipline, but if you want to be a successful project manager you have to stop thinking in this code-centric fashion. Instead, you should see it as just another stage, neither more nor less important than the others. Remember that if the requirements are wrong, the best programmers in the world won't deliver anything useful, and that until it's tested, all software is unproven and therefore of little value.

As well as developing this discipline for yourself, you have to develop it in the members of your team. Most programmers are at their happiest writing code, but you need to create an environment in which the delivery of the complete package (code, documents and documented tests), to a known quality standard, is what counts, not the code alone.

Be prepared, as well, to recognise the strengths and weaknesses of your team members. If some are less good as programmers, it may advance your project more rapidly if they concentrate on non-programming tasks, leaving the code-cutting to those who are best at it. Alternatively, you may find that your best programmers are also the best testers and documenters, and you have to make sure they do their share of the documentation and testing.

To make this work you will have to elevate the importance of the documentation and testing to at least equal with the coding, and possibly higher. Nobody likes to think they are second best, and the truth is that if they are doing these other jobs they may be contributing more, not less, to the project.

You'll also have to try to make the documenting and testing jobs more rewarding. This means making sure that these jobs have well-defined objectives, and are properly planned and resourced. If you've got specialist documenters or testers, then dedicate them to the appropriate jobs but recognise the special contributions they are making.

There is one further benefit of this approach. If you have resources concentrating on documentation and testing at the same time as the coding, you're less likely to be caught in the "testing trap" with a pressing deadline and these important jobs not done.

What Do I Do?

1. As project manager, you must:

   - Lead and build the team,
   - Plan,
   - Communicate,
   - Monitor and report progress,
   - Achieve the overall project objectives.

2. A plan is only complete when for each job you can answer the questions *what?*, *why?*, *when?*, *where?*, *how?* and *who?*.

3. Talk to people, make sure they understand what they are supposed to do, and what other people are doing or expecting of them.

4. Don't confuse your project management tasks with any technical responsibilities you may have. You must dedicate some time to the management role.

5. The key aspect of leadership is to focus everyone on a common goal. Cooperation is better than coercion.

6. You don't have to give orders to get people to do things. If you understand what motivates each person, there are much better ways.

7. Don't be afraid to delegate.

8. Accept that mistakes happen. Build an environment in which there is a positive effort to find errors, correct them and prevent recurrence, but not as a criticism of the person who made the error. Submit your work to the same checks.

9. You can't force a team to "jell", but you can create the right environment by creating an open framework, avoiding defensive management and excessive paperwork and paying attention to the working environment.

10. There are a number of key roles in any team. Check your team to make sure that all the key players exist.

11. Concentrate on making communication effective. Remember that you may need to communicate important messages in more than one way. Communicate in all directions: up to your managers,

down to your team and out to the users.

12. Learn how to sell your ideas to your managers. If you have a problem introducing the right methods, look carefully at whether your organisation is ready for such changes. If not, concentrate on the most basic good project management practices.
13. You don’t have to be an expert, as long as you can ask the right questions, and find ways of making people explain what they’re telling you.
14. Look for problems before they occur, and think of ways of preventing them, or, if all else fails, bringing in a contingency plan to limit their effect. Formal risk analysis is a very good way of doing this.
15. Programming is not the most important activity in a software development project. You have to make sure that you expend good-quality effort on the management jobs, and on things like documentation and testing.
16. Get a firm sign-off of each stage or interim deliverable.

📖The best further reading in this area is, without a doubt, *Peopleware*. *Make It So* and *Debugging the Development Process* are strong on how to direct and manage people. *The Decline and Fall of the American Programmer* and *Software Project Management* also have very strong “peopleware” sections, including good advice on how to build effective and balanced teams. The latter book goes into a lot more detail on risk analysis.