Career Development for Development Geeks.

(c) 2006 Erik de Castro Lopo

erikd@mega-nerd.com

1 Introduction

Software engineers working as employees are notoriously bad at managing their own careers. They are often underpaid, overworked and under-appreciated. This paper will look at some of the causes of these problems and look at what developers can do to improve their lot. Many of the proposed tactics can be initiated by single developers or development teams to improve their work environment independent of their organization's management.

2 Understanding the Problem

Many software engineers are faced with work environments where they routinely work 50, 60 or more hours a week for stretches of time ranging from a week to months. They work for managers who set unrealistic and often totally arbitrary deadlines; work on projects that are poorly planned, poorly executed and always behind schedule. This leads developers to be seen (quite possibly unfairly) as being unreliable. Developers that are seen to be unreliable are unlikely to get the pay rises or the appreciation that they may deserve.

3 Long Work Hours

Any developer who regularly works more than 40 hours a week needs to take a good look at their work practices. Here are some factors to consider:

- How many of their total hours are actually effective? Few developers can do much more than about 40 hours of highly productive in a week. Hours spent at work when the developer is not working effectively is time wasted for both the developer and the employer. Long hours makes developers tired and doing it week after week means they never get to recover.
- What do long hours tell the employer about the employee? If a manager has two developers; one who spends 40 hours a week at work and mostly meets their deadlines and another who needs to work long hours when a deadline approaches and then often runs overtime; which developer looks most competent to the manager?
- Developers under pressure will often discard the good programming practices (like refactoring, testing and commenting) that they would normally use if not under pressure. Discarding good programming practices only exacerbates the problem.

Situations like this are a problem, so why do (some) developers do it?

In many cases its a matter of the developer trying to catch up to where they think they should be at on some project plan. In other situations its a problem of trying to meet unrealistic deadlines that have been imposed on them combined with management's use Microsoft Project's Gantt charts.

4 Why does this Problem Exist

Unfortunately, many developers who face these problems blame their managers. This is very rarely the right place to lay the blame. It may seem a controversial proposition for some, but situations where a developer is working under these conditions, it is usually the fault of the developer. This is not a case of blaming the victim; in this situation, the person suffering under these conditions has much more control over the situation than they think they have. The up side is that developers faced with a work environment like this has the power to change it.

5 Steps to Salvation

The steps to overcome these problems are:

- Developers need to manage their managers.
- Developers need to manage their projects.
- Developers need to measure their development performance.
- Developers need to apply engineering principles to management of the above.

6 Managing Management

Any and all developers that have managers need to be aware that they need to manage those managers. Tradition seems to suggest that managers lead and developers follow; that developers are subservient to managers. Unfortunately this model doesn't work very well for a task as complex as software development.

Instead, the relationship between managers and software developers should be more like a partnership, with the engineering manager acting as the interface between the software development team and the rest of the organization.

One of the most important aspects of interacting with a manager is managing their expectations. If the developer can't given them a clear indication of how long a task is going to take then they don't have enough information to provide development time-lines to their managers.

Managing the expectations of management is almost totally down to the developer. When a manager sets a developer a task the developer needs to be able to estimate how long the task is going to take. This requires project planning and estimation skills; something that is rarely taught in software engineering courses.

7 Managing Projects

Unrealistic deadlines and Gantt charts are management's reactions to uncertainly in development projects. If the developer or development team can can show management a developer centric project plan then most managers will just incorporate the developer's time estimates unchanged into their own estimate that they supply to their management. With a well document estimate coming from the developers, the manager has no reason to to modify it.

Once the project is under way, the developer must provide a steady stream of progress reports that reference the original plan. This will allow management to make allowances for any project overrun much earlier than they would otherwise have been able to do so.

In the real world, we know that project requirements can change as the project is underway. When this happens, the developer needs to update the project plan as soon as possible and feed the changes in time estimates back to management as soon as possible.

Once a developer starts using the techniques described here, they should learn from their mistakes and try to improve their project management and estimation skills. This means recording the differences between the estimate and the actual time taken and then trying to correct for those mistakes next time.

8 Project Estimation

Project Estimation is the part of software engineering that very few software developers even consider part of their job description. Many consider this a management task; and those developers will have Gantt charts and unrealistic deadlines imposed upon them. Fortunately, the kind of project estimation required is actually rather easy and isn't very time consuming either. Estimating the time on a two month project should not take more than about 4 hours and keeping the project plan up to date and reporting progress to management should not take more than an hour a week. Considering that the costs are so low, no developer can afford not to do project estimation on the projects they are involved in.

The steps required for estimating a time line for a project are as follows:

- State the project goals.
- List all the tasks required to complete the project goals.
- Estimate the time for each project task.
- No task should be less than half a day or more than 3 days. Those longer than 3 days should be split into sub tasks and estimated separately.
- Testing tasks should be included. Possibly also documentation.
- If the whole project is longer than say 3 months, split it into shorter sub projects and estimate them separately.
- Mark the tasks with the highest risk and attempt to schedule them early int the development process.
- Add up all the times.

- Do not multiply the total by 2.
- Do not add a fudge factor.
- Do not be overly optimistic.

Once an estimate has been made, submit it to management and don't let them fiddle with it.

9 Working the Project

During the course of working the project, the engineer should, on a weekly basis or when milestones are reached, give management updates comparing progress so far with the estimate. Its also important to point out where interruptions caused delays. If two days are spent doing something un-related to the project, management needs to know this so they can decide what is most important; the project or the interruptions.

Even with the best estimates in the world, projects can still go off the rails. When this happens, inform management about the problem, possible solutions and redo the estimate. Recalculating the time estimate should be a rare occurrence arising from situations which could not reasonably have been foreseen.

10 Measuring Performance

The project estimation techniques described above gives a developer's manager important feedback on their performance. The beauty of this is that the developer has a large amount of control over how manager perceives their performance. It follows from this that a developer that is perceived to be performing better than his or her colleagues is of more value to the employer and their salary should be related to the value they provide.

Another factor which can influence how performance perceived is the company bug tracker. If the employer has a bug tracker which assigns bugs to individuals, its usually worthwhile for a developer to try to keep the number of bugs assigned to them and their time to resolution to a minimum. A good time to tackle these bugs is when the developer is running slightly ahead of schedule and has some slack time.

Its also good for a developer to keep their own personal record of wins and losses. Likewise, when a developer has a win, he or she should let their manager know.

Measuring performance leads us to the issue of performance reviews. Before a performance review, a developer should read over their list of wins and losses. They should also look over how effective their project planning was and how close their time estimates were in comparison to how long the project actually took. During the performance review let management remember the failures but make sure to remind them of the successes. It can also be useful to ask management how performance is measured. If a developer knows how performance is measured they can tune the way they do their job to maximize their perceived performance.

11 Managing a Long Term Career

Software engineering is often seen as a young person's game. Many older software engineers and developers feel that is they want career advancement they have to move into management. Those that wish to resist moving into management need to make themselves more valuable than younger engineers.

The best way to do this is provide employers with the one thing that younger engineers will have difficulty providing; experience. Experience cannot be taught in a university course, it can only be gained by making mistakes and learning from the experience.

Other factors which allow experienced engineers to improve their long term career opportunities include:

- Diversification of skills.
- Eagerly learning new skills and fields.
- Refusal to be pigeon-holed.
- Taking on new challenges with gusto.
- Moving from field to field; preferably from low demand fields to high demand fields.

12 When all Else Fails

If the project planning, good project estimates and constant manager feedback fail to achieve the desired results, then its probably time to look for a new position. Unfortunately, hunting for a new position is outside the scope of this paper. However, the techniques discussed above work even better for new employers than they do for existing employers.

13 Conclusion

This paper shows how software developers do not have to suffer under long hours and oppressive development schedules. By taking charge of their projects and working with their managers they can turn their working lives around so that they enjoy a less stressful work life, sane working hours, projects that are under control and gain the respect of their colleagues and management.