9 Keys to Effectively Managing Software Projects

Introduction

Can managing software development be as simple as reading a brief "to-do/not-to-do" list? No. All evidence indicates that software development is especially difficult to manage. If not, why do repeated studies come to these same conclusions that most software projects are

- over budget
- do not meet their schedules
- do not fulfill customer requirements

The 9 Keys address issues that are fundamental to all software development and the environment in which it is developed. More than a "to-do" list, they provide the busy business leader with a framework for high-level planning and monitoring. There is no magic here; only guidance about what your expectations should be, what you should monitor, and how you should respond.

Key #1: Software Development is not Manufacturing

The modern corporation evolved in a manufacturing environment. If you wanted to increase production you added another assembly line, a second shift, or increased machine speed to reduce assembly time. It was relatively easy to quantify the outcomes. You knew what to do and how to implement it. In contrast, software is often a discovery process. Initial requirements are at a functional rather than an implementation level. The higher the level of the requirements the more time and effort is spent determining what to do and how to do it. Adding additional staff (a second shift in the manufacturing metaphor) will contribute little to defining what needs to be done and how; but will increase the cost and degrade the quality of what finally is delivered. The more abstract the requirements, the newer the development tools, the more unfamiliar the development team is with the business and the environment, the greater the time and effort required to determine what to do and how to do it.

Key #2: Schedule and Cost are not Interchangeable

The relationship between schedule and the cost/effort required to complete a software project is profoundly non-linear. For any project there exist optimal schedule and staffing ranges. When a schedule is shortened to meet a deadline, the team will not function as efficiently. The quality of the delivered product will also be degraded due to the increased communication complexity and reduced testing. Moreover, there exist very real limits to the amount of schedule compression that is possible. Unrealistic schedules are the number one cause of project failure. Nearly 30 years ago, Frederick Brooks noted

in “The Mythical Manmonth” that adding staff to a project that was behind schedule only helped it to be further behind schedule. Figure 1 illustrates the schedule/effort tradeoff. For any software project there is a continuum of feasible solutions that is framed by what is impractical and what is impossible. Note that as the schedule is reduced the effort (cost) increases exponentially.

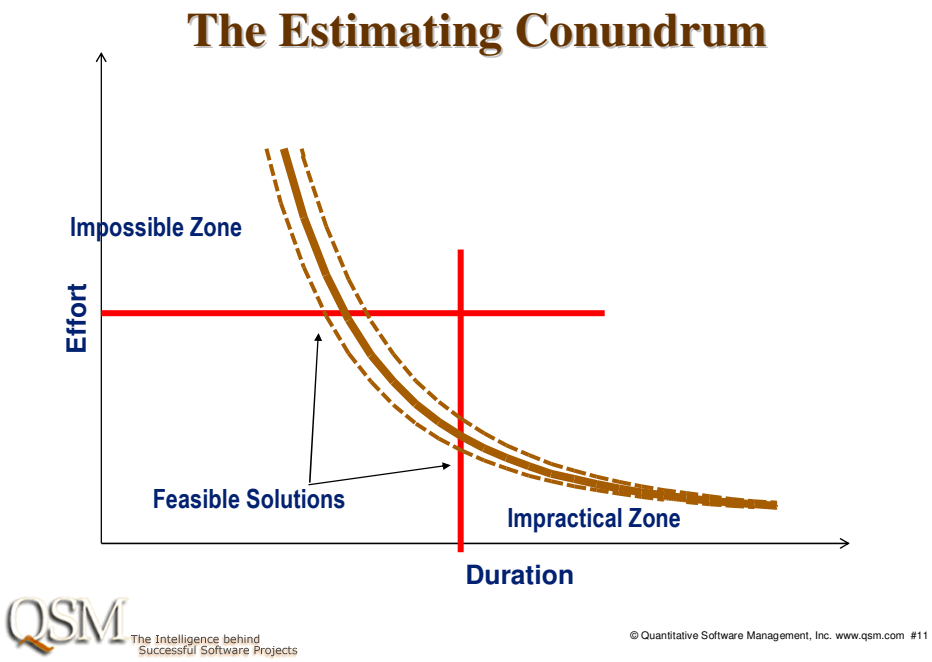

Figure 1

Key #3: The Best Data for Comparison is Your Own

For all of the time, money, prestige, and effort that are tied up in a software development project, it is nothing short of scandalous that many companies do not record, analyze, and use the schedule, cost, and quality data that all projects generate. This data forms a composite picture of how an organization develops software. With it, strengths and weaknesses can be identified, estimates can be grounded in reality, and achievable targets set for improvement. What you measure you can rationally manage; what you don’t measure … . Capturing summary information for each project: how long it took, how much effort was expended, how many defects were recorded during development, and what it produced should be a task in project close-down for every project.

“Industry” data can be useful for comparisons; but it cannot tell you about your organization and how you produce software. Most organizations have a very discernible pattern to the way they produce software; but if project history is not kept, very valuable information about your organization remains unknown and you will not know if a

particular schedule is feasible, a staffing plan appropriate, nor whether a quality standard is attainable. KEEP AND USE YOUR PROJECT HISTORY.

Key #4: Don't Get Lost in the Details

When initially planning a software project it is important to focus on the large questions

- Do we know enough to make an intelligent estimate?
- Are the project's schedule, cost, and quality constraints achievable?
- Can we staff this project?
- Is there an agreed upon change control process in place?

Remember, most projects are over budget, exceed their schedules, and don't meet customer expectations. This is because they did not address the big questions that focus on feasibility before addressing the minutia. Every project needs a project plan with roles and responsibilities and a critical path. But, the project's overriding constraints must be honestly addressed before detailed planning begins.

The same holds true when collecting project data. Capture only what you need and can use. Make sure that it is summary data and be as unobtrusive as possible: developers don't enjoy providing data and will provide bad data if overwhelmed by requests.

Key #5: Change Doesn't Come Free

Software projects are a delicate balance of tasks to complete, staff to work on those tasks, and time in which that work is done. A change to any one of these affects the other two. As a rule, changes to requirements (the tasks to complete) in the latter stages of development are the most costly since they may require extensive rework of already competed tasks. Except in very small projects it is normally not practical to freeze the requirements so that there is no change. As we noted in Key #1, software development is a discovery process. There are two ways to handle change effectively that should be used in tandem. First, since you know that there will be a certain amount of change in a project, make allowances for it in both budget and schedule. A sure path to failure in software development is to plan on a best case scenario which is really an exercise in self deception. A realistic project plan has room in it for some false starts and unforeseen events. Good project management anticipates as many of these as possible and plans how to deal with those it cannot foresee. Second, in conjunction with this, every change should be evaluated for its impact on cost and schedule which will need to be adjusted if the proposed change is accepted. In summary, plan for change and evaluate its impact.

Key #6: Plan and Monitor your Plan

Every project should have a project plan that specifies

- What will be delivered
- Staffing
- Schedule

The plan should be based on historical performance so that commitments are realistic and achievable. In addition, the plan should contain critical milestones to measure ongoing performance.

Most projects have plans, so why do so many fail to meet their cost, schedule, and quality objectives? The reasons are many. If an organization does not keep and use its historical project data it does not know what its capabilities are and may make commitments it cannot fulfill. Another problem is the failure to monitor accurately a project when it is in process. When a project begins to miss its milestones there is an all too human tendency to try to make it up in future reporting periods. This rarely works. Its actual effect is to hide bad news and postpone its delivery until it is too late to deal with it effectively. The best indicator of how much a project will cost and how long it will take to complete is its actual performance to date. Commercial forecasting tools, like SLIM-Control®, can do this. Nobody likes to be in the position of delivering bad news; but, bad news is best delivered sooner, when there may be alternatives to consider, than later when there are none.

Key #7: An Estimate is not a Project Plan

When estimating a software project, there is always some uncertainty about two key components, the "size" of the project requirements and the productivity of the development team. There are many good sizing techniques that are useful in estimating size; but there will always be some uncertainty around it. Remember, part of software development is determining what to do and how to do it. Likewise, productivity can be impacted by the staffing strategy selected, schedule constraints, the team's familiarity with the application and technology, and by the organization's process maturity. So, there simply is not sufficient information available to say, "This project will last 10 months and cost $750,000." 10 months and $750,000 may be the most likely outcome; but there is insufficient information available to predict this exactly. What an estimate can do is quantify the uncertainty around cost and schedule. This can be done several ways. One common way is to frame the estimate with best case, worst case, and most likely scenarios. Another option is to estimate to a particular level of certainty, "This project has an 80% chance of completing in 11 months for less than $800,000". Estimating software such as SLIM-Estimate™ can do this very easily. Framing an estimate is particularly important since the staffing, schedule, and cost numbers it presents create performance expectations.

In summary, an estimate provides the big picture of a project and the level of risk for any particular solution. A project plan provides the detail of that project: what the individual tasks are, roles and responsibilities, and the critical path. Estimates and project plans are complimentary.

Key #8: There's a Lot More to Software Development than Coding

How important is the choice of software language in determining a project's productivity (i.e. its ability to meet schedule, cost, and staffing constraints)? QSM is frequently asked to evaluate the impact of using one language as opposed to another and the answer appears to be "Not much".[1] There are several reasons for this. First, the skill level of the team with the language may be more important than the innate capabilities of the language: especially if several languages are appropriate for the task at hand. Second, the majority of work in a software development is not in the coding. Requirements gathering, analysis, design, project management, configuration management, documentation, system and integration testing typically require 70% of a project's total effort. So, even if you could reduce the effort involved in coding by 50%, that would only reduce overall effort by 15% and have an unknown impact on schedule. The languages use for development should be appropriate for the task and be a good match to the team's skills. They are not, however, the silver bullet that will solve all of your problems. Good project management, change control, and configuration management will do more to improve a project's overall performance.

Key #9: Plan for Project Measurement

Every company that develops software would like to know whether it is more or less efficient that its competitors, whether the accuracy of its estimates is improving, and what is normal performance for quality and productivity. Why, then, do so few companies have this information? After all, the data needed to determine these is a normal by-product of every software project. There are many reasons for this; but they ultimately boil down to two fundamental ones: the information is either not collected or it is collected so inconsistently that comparisons and trends are impossible. If no data is being collected, leadership has not required it. If it is collected inconsistently, standards have not been established and enforced, or they are so unreasonable and burdensome that they are circumvented.

Software measurement and analysis don't just happen. Like any business initiative, they need clear objectives, senior leadership support, established roles and responsibilities, standards, and knowledgeable staff. By itself, project measurement does not fix anything. If your productivity and quality are poor, measuring them does not remedy the problem. However, measuring them will tell what they are so that you have a quantitative basis for evaluating the effectiveness of your improvement strategies.

Software measurement is a powerful tool for planning, managing, and evaluating software development and maintenance. When software is a key component is a business' success, knowledge of your capabilities may be the difference between profit and loss, success or failure.

---

[1] "The QSM Software Almanac, Application Development Series", 2006, p50-51. The chapter, "Best in Class, Worst in Class" examined productivity and was unable to identify any relationship between language and productivity.

Conclusion

QSM is the oldest software metrics consultancy in the world. If you would like to comment on this article, please send us an e-mail at info@qsm.com. If you need help in deciding the how's and what's of establishing a successful software measurement program, QSM's Project Office Setup service may be just what you need. Avoid the pitfalls and take advantage of our decades of experience. Contact us a 1-800-424-6744 or at info@qsm.com.