Set the Stage for Success

Troubled software projects quickly devolve into a blame game. Management blames the project manager. The project manager blames the developers (and, secretly, management). Developers have a wide array of blame options: senior management, project leaders, suppliers, customers, partners, sales — even each other. More important than assigning responsibility is determining what measures can be taken to reduce the incidence of software projects that miss their schedule, exceed their budget, deliver a mediocre product, or any combination of these.

The fact is management decisions made before the project is underway are a significant determining factor on whether a software project succeeds or fails. Management choices can either handicap a project before it begins or create the environment in which it can succeed.

Software development as a business or practice is 50 or 60 years old and ample data exist to distinguish between measures that promote success from those that contribute to failure. While adhering to the following seven principles will not guarantee that every software project will be successful, they will reduce the incidence of problem projects. As a corollary, ignoring them practically guarantees failure.

1. Your project is bigger than you think

This is a perception issue and can be very subtle. Simply stated, no matter how thoroughly the proposed software is analyzed, there remain issues that will only surface as the actual work is being done. If this were not the case, if software development were a straight forward deductive process, there would be no need for developers at all. We would simply feed the results of our analysis and design into a Case tool which would generate the code to support the logic. Unfortunately, Case tools did not prove to be a panacea and did little to improve overall productivity. The track record is that software projects take longer to develop, require more effort (i.e. cost more), and create more functionality than were originally planned. A study conducted by QSM found that on average projects exceed their schedules by 8%, cost 16% more than planned, and develop 15% more functionality than anticipated. Software consultant Capers Jones has stated that projects grow on average 1.5% per month.

Skilled project managers intuitively recognize that there are unknown factors and try to account for these by buffering the budget and schedule in their project plans. Often, these are the first things to be cut during plan review since they cannot be identified and the project plan becomes a best case scenario: one that seldom plays out in reality.

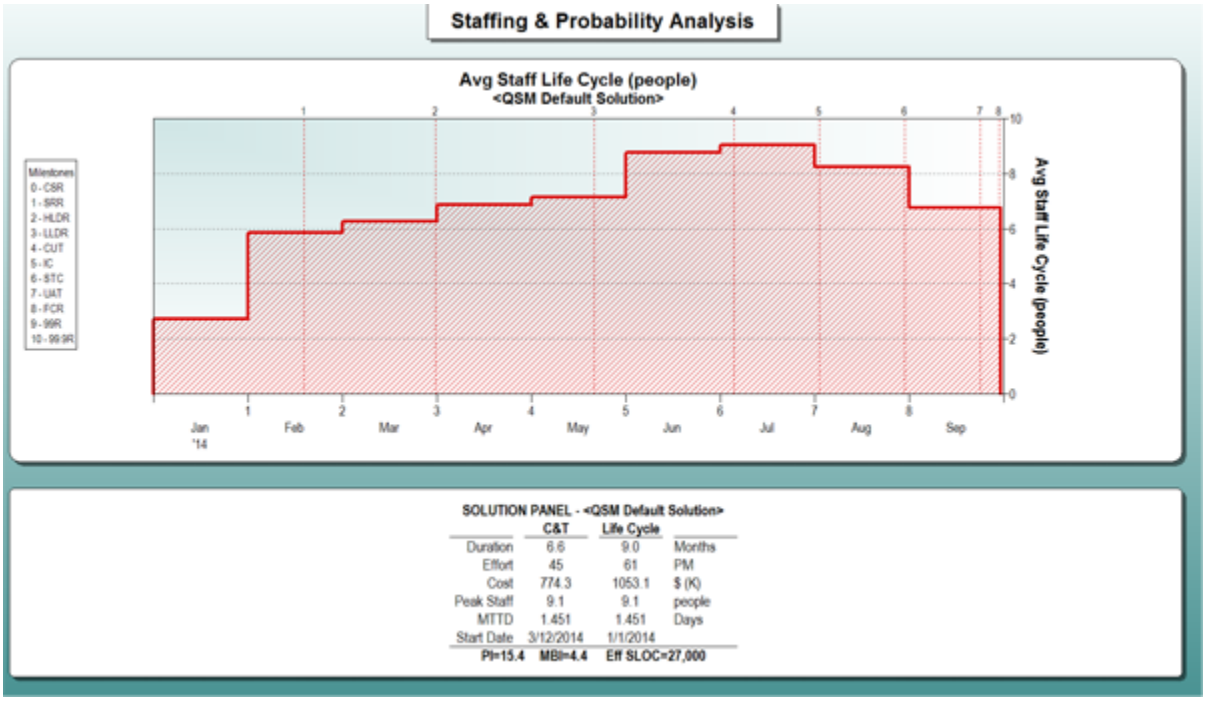
This is a point at which business leaders can make a critical and positive contribution to project success. If the software is important enough to the enterprise to develop, make sure that it has sufficient schedule and budget to succeed. Since it contains more functionality than can be seen, and will require more time and effort to be developed, plan for this to happen. This is emphatically not padding an estimate; it is recognizing reality and planning accordingly. If you are using a parametric estimation tool to determine budget and schedule, increasing the size of the functionality to be developed by 15% will assist you in your planning. Remember, the additional cost, schedule, and functionality are already there. You just can’t see them. Refuse to acknowledge their existence and the project is well on its way to exceeding both budget and schedule.

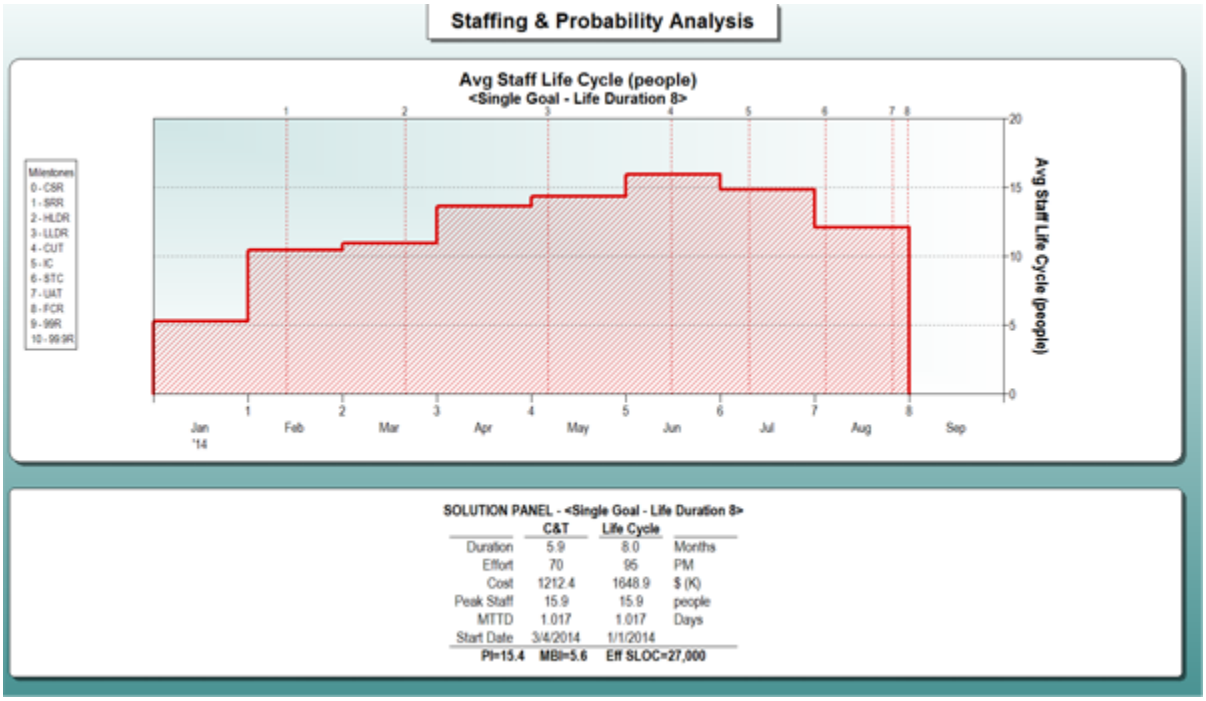
2. Schedule and cost/effort are not interchangeable

Our concept of productivity comes from manufacturing. However, there is a key difference between software

development and manufacturing. In a factory, if you have one assembly line and want to double production or reduce by half the time required to produce the same quantity, you add a second assembly line or a second shift. In essence, you double the effort to double your output or reduce the schedule by 50%. The relationship is linear.

This does not work for software development. In software, the relationship between schedule and effort is non-linear: one unit of schedule reduction is purchased at the cost of many additional units of effort. This is not theory; it is supported by 40 years of research. Figures 1 and 2, below, illustrate this concept. The top chart represents a business financial project that lasts 9 months and is average for productivity, effort, and schedule when compared to the QSM database. The bottom chart illustrates what occurs when the project is forced to complete in 8 months.





In essence an 11% schedule reduction is purchased by a 56% increase in cost/effort. It is beyond the scope of this paper to go into detail why this occurs; but the fact remains that it is the wisdom derived from thousands of completed software projects. Neither of the solutions illustrated above is impossible; both are well within the normal range of variability observed for projects. Business leaders concerned with spending their IT budget judiciously can accomplish a great deal simply by relaxing the schedule modestly. (This will also have a positive impact on product quality, since testing won't have to be shortchanged to meet an arbitrary end date, and may improve team morale which could reduce staff turnover.) It is important to note that the relaxed schedule needs to be planned into the project, not added when the project is about to miss its scheduled end date. Planning a more relaxed schedule allows the project to take advantage of the non-linear relationship between schedule and cost/effort and use a smaller and less expensive development team.

3. Know your capabilities

Organizations are creatures of habit. What this signifies for software is that within a company or a company division, there are identifiable patterns for how projects are staffed, their productivity, quality, and whether schedule or cost is optimized. These patterns are not immutable; but they cannot be changed until they are identified and effort is consciously directed to alter them.

Do you know what your organization's patterns are? If not, the old adage applies that those who ignore history are condemned to repeat it (and any haphazard improvement efforts that are attempted will flounder). Every software project has the following features:

- It creates something (which can be quantified)
- It requires time and effort to do this and has an associated cost
- Issues arise throughout this process (defects)

These form the basis of the data that should be collected for every software project. With them productivity, schedule, and quality analysis can be done quantitatively. Impossible schedules (for upcoming projects) can be identified. Tradeoff analysis between cost and schedule becomes possible (and empirically based). Are you collecting these as every software project completes? Are they maintained in a database where they can be analyzed? These data are the byproduct of all software projects: size, schedule, effort, and defects. If you are not currently collecting and analyzing these, this is where you need to begin.

4. Keep the team small

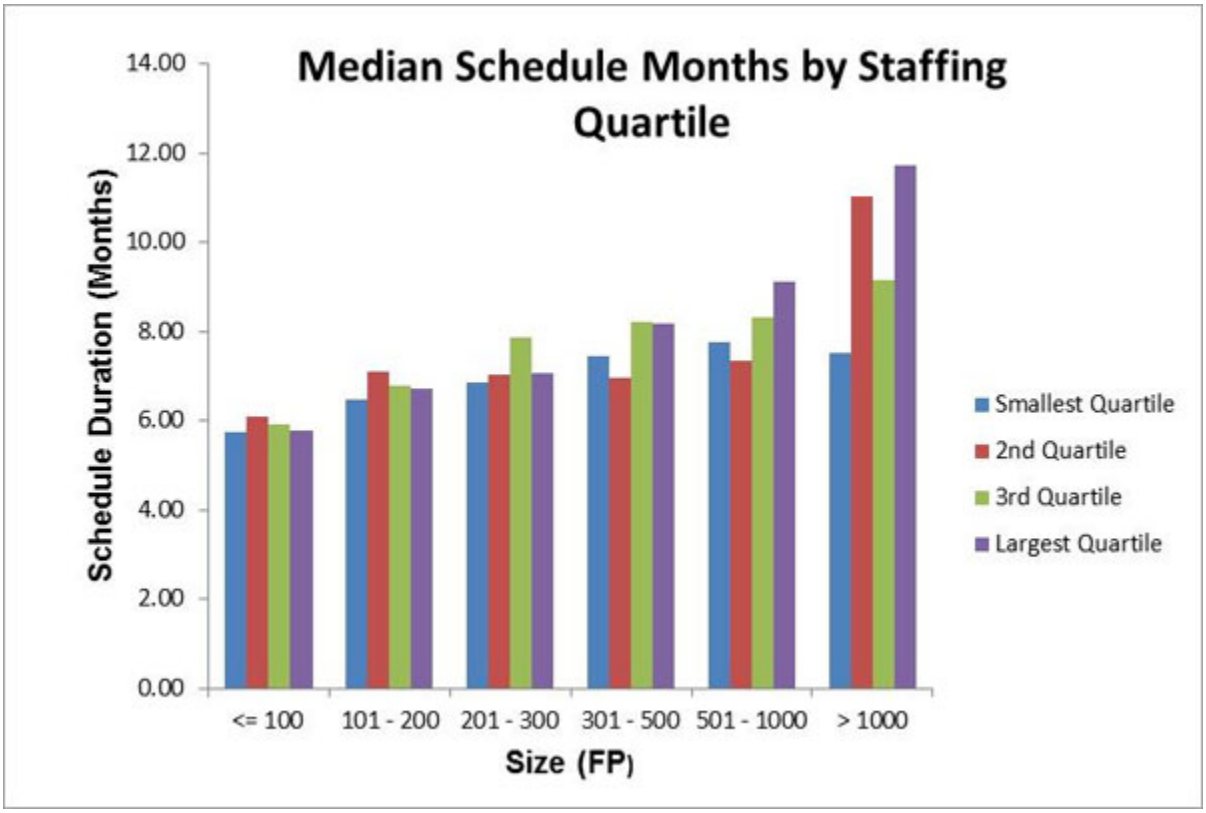
There are compelling reasons to staff software projects sparingly:

- It increases productivity – significantly
- It *does not* affect the schedule (make projects take longer to complete)
- It costs less

Here is the supporting evidence taken from a study of over 2,100 completed software projects. In the following table, the projects were sorted into size categories. Each size category was then sorted by the average project staff. For each staffing quartile within a size category the median (and average) productivity was calculated. A comparison was done between the productivities of the highest and lowest staffing quartiles for each size category. Without exception, the lowest staffing quartile was more productive with a range between 277% and 804%.

Productivity Rates (FP/PM) Smallest to Largest Staffing Quartiles					
Size Range (FP)	Lowest Staffing Quartile		Highest Staffing Quartile		Productivity Ratio
	Productivity (FP/PM)	Median Staff (FTE)	Productivity (FP/PM)	Median Staff (FTE)	
1-100	7.17	0.86	2.57	2.53	2.77 to 1
101-200	13.68	1.19	2.83	4.41	4.83 to 1
201-300	17.44	1.59	3.15	6.62	5.54 to 1
301-500	27.15	1.73	3.96	7.47	6.86 to 1
501-1000	34.96	1.76	4.35	10.95	8.04 to 1
>1000	45.29	2.86	5.76	15.04	7.86 to 1

The most frequent rejoinder to information like this is “I don’t have time to use a small team. I’ve got a deadline to meet”. The following figure graphs the median schedule for each quartile for every size category and shows that larger team sizes do not complete projects sooner.



Deadlines are real; attempting to meet them by using a large staff is ineffective, and for larger projects, counterproductive.

5. Get an independent evaluation of your plans

The process of determining the schedule and budget for an upcoming software project may be subjected to intense pressure from persons or groups with conflicting interests and opinions. What comes out of this process may or may not be practical or even possible. In most cases the result is not empirically based. This is where an independent assessment by an experienced estimator using a parametric estimation tool becomes extremely valuable. As an “outsider” with no skin in the game, this person can generate likely outcomes that can be compared to the desired ones.

If your organization has been collecting project history, the estimator’s models can be based on proven capabilities. A project with an inaccurate budget, schedule, or staffing plan is well on its way to failure before it gets underway. Getting an independent estimate from a person or team whose job is to produce good estimates

will identify potential problems before they manifest. By insisting that independent tool based estimates be done for upcoming projects, business leaders can help avoid potential pitfalls and help create a plan that can succeed.

6. Track performance to plan and re-plan as needed

There is an old saying that no battle plan survives first contact with the enemy. Software projects share this trait and their plans must be able to adapt to what is transpiring. Attempting to keep to the original budget/schedule plan, when the assumptions it was based on are no longer valid, flies in the face of reality.

Where leadership can make a positive difference here is by establishing sound processes for monitoring projects. This has traditionally been done using financial measures. Unfortunately, by the time financial monitoring indicates that a project is in trouble, it is usually too late to take effective corrective action.

There is a better way. Every project plan has a schedule with milestones and a plan for expending effort (and money). Deliverables are created and tested. Using a parametric monitoring tool, these metrics (milestones, effort, software deliverables) can be used to track progress and forecast likely outcomes. The advantage of these over strictly financial monitoring is that this can be done much earlier in the project when there may still be time to consider alternatives and take corrective action.

Identifying a problem does not resolve it. But, knowing that it exists sooner rather than later is a clear advantage and may allow alternative possibilities to be explored.

7. Allow time for planning

A consequence of short schedules is that both planning and testing do not receive the time and effort that they require. The results aren't pretty. Inadequate planning leads to software that does not meet the business requirements. Abbreviated testing allows defects to be discovered in production where they are more expensive to fix and visible to customers. QSM has twice conducted studies that compared projects that spent more than average effort for analysis and design (planning) with those that used less than average. Both times, the results were striking. Projects that spent above average effort in analysis and design completed sooner; had higher productivity; used less effort (cost less); and produced fewer defects. The table below summarizes the results for the more recent study An Analysis of Function Point Trends:

Comparison at 20% Design Effort		
Medians		**% Difference**
PI <= 20%	**11.04**	
PI > 20%	**14.19**	**29%**
FP/PM <= 20%	**6.20**	
FP/PM > 20%	**7.93**	**28%**
Duration <= 20%	**7.23**	
Duration >20%	**6.20**	**-17%**
Total Effort <=20%	**22.59**	
Total Effort > 20%	**20.29**	**-11%**
Average staff <= 20%	**2.34**	
Average staff > 20%	**2.50**	**7%**
FP size <= 20%	**157.00**	
FP size > 20%	**171.00**	**9%**
Defects <= 20%	**20.00**	
Defects > 20%	**19.50**	**-3%**

This is another area is which business leaders can exercise a positive influence on software development by insisting that sufficient time and effort are allocated to planning. The projects will complete sooner, cost less, and have fewer defects.

Bringing It All Together

While business leaders do not directly manage software projects, they exercise a profound influence on them through the decisions they make. In summary, here are the decisions leaders can make that create the environment in which projects can succeed.

- Plan for growth. Projects are larger than they seem at the outset and will require more time and effort to complete.
- Give projects the time they need to succeed. Setting an aggressive schedule is the single worst thing that can be done to a project.
- Collect and use project history to make empirical decisions.
- Use the smallest staff possible to complete the task. It won't hurt the schedule and costs a lot less.
- Get an independent evaluation (estimate) of every project – and pay attention to them.
- Establish and enforce procedures for ongoing monitoring of software projects.
- Spend time and effort up front determining what to do before starting to do it.