On Time, In Budget – Challenges of IS Manager

One of the most challenging but often under-estimated task is that of managing an IT (software development) project. It is disheartening to see that the almost everyone assumes that software projects will overrun their budget and will always be delayed, no matter how much money or resources you throw at them. Thanks to Microsoft delaying the releases of its most publicly visible software, Windows, that this assumption has become an industry norm.

Over the course of my career I have come up with a list of 'Failure Seeding Factors' that most commonly influence the success or failure of a software project. In decreasing order of significance, these are –

- Scope creep
- User involvement or lack thereof
- Incorrect estimation & planning
- Technology overload
- Incorrect prioritization
- Underestimating the 'human' factor

The reason I call these *seeding* factors is because their effect is not visible at the time they are introduced into the system, but well after it. For example, the scope creep begins right at the very first design meeting, slowly creeping into the system resulting in a big impact on project schedule or cost even before the design is completed!

Scope Creep

One of the most common, difficult to control and subtle factors in project cost or duration overrun is scope creep. In simple words, scope creep means that you started out with a plan to do task 'A' but you ended up doing task 'A+B', and surprisingly it may start with the very first design meeting!

This happens because of multiple reasons, the most important being - lack of proper documentation at each stage of the project. If the team that does the scoping and analysis for the project clearly outlines the inclusion and the exclusion list, project scope can be controlled. Normally, we focus on what will be included in the system but do not explicitly list what will *not* be included. This leads to a situation where coupled (e.g printing & sorting) functionality creeps in with the assumption that someone must have overlooked it.

To control scope creep, a change control process should be defined and adhered to for any type of changes – business functionality or technical. It also helps to document minutes of the meetings with decisions highlighted for later reference. The ownership of taking decision about the changes should be appropriately centralized without creating bottlenecks.

User Involvement or Lack thereof

User involvement is a double-edged sword. I have come across projects that failed because users were not involved and also the ones that failed because users *were* involved.

Though it is not always true that early user involvement can guarantee the success of a project; but it is almost always true the other way around. On the flip side – if the user champion is not knowledgeable enough or is a poor decision-maker the project is at a greater risk of failure. In this case you may have to train the user or ask the user group to select another champion.

Not involving the users at the right time creates us vs. them feeling right from the start of the project. The users view the development team members as geeks who are as aware of the business needs as a fish is of a bicycle, and the development team thinks of all the users as morons – why else would you want to build a 'fool proof' system! Think of users as champions for your project in side your organization and with higher management. If the trust relationship is built between the technical and business team (user champions) right up front, then each team will be more tolerant about each others needs and problems and will go the extra mile to accommodate.

Incorrect Estimation & Planning

This factor is the main contributor to the budget overruns of a project. Even though a lot of blame goes to the project manager and team leader for not being diligent enough, some of the blame also goes to the developers.

When a developer is asked how much time will be taken to develop any functionality, more often than not you will get an estimate for the actual coding time and not *development* time. This is a universal problem as the majority of developers assume – if it compiles, I'm done. If you add technical documentation, unit test, rework and reviews/QA time the estimates easily double in size. This also indicates why most of the IS projects lack in documentation, if they have any, or are not production worthy when they are rolled out.

Adding to the problem, managers seldom account for vacations, sick time, holidays or mid-term departure of a developer when preparing a project plan. Given the mobile nature and job prospects of software industry a developer may leave without enough notice!

One way to circumvent the problem is to show each of the development activities as separate line items on the project plan. This way the reader will get a clear picture on how much effort will be spent on coding, unit testing and documentation, etc. It also will re-enforce with the developers that coding is only one of the tasks in development and other tasks need to be completed in order to produce a quality product.

Technology Overload

Being from a technical background and hands on manager, this is one of my favorite factors as I can personally relate to it.

With few exceptions, most of the development is done with latest tools and technologies. Even as the tools get older, their relentless version upgrades and fixes or patches provide the thrill in a developer's life. The developers, in order to stay current with changes in technology, try to use every (and I mean *every*) feature of the software somehow or the other in their modules. If for no other reason, then just to 'check them out'. This creates an overload on the rest of the team, as they will have to jump through hoops to make their pieces work together, which further complicates testing, maintenance etc. In order to control this overload plan on code reviews at regular intervals in the project and weed out all unnecessary code.

Another related but incorrect assumption I have come across is – developer time is more expensive than user time. This results in developers getting the fastest and loaded desktops that are at least four to five times faster than the target configuration. The software being developed will meet or exceed the expected performance on developer machine, during various demos, but will run dog slow on the users' machines. Because the developer will have no bottleneck in performance s/he will not understand the importance of 'optimizing' the code.

In the end, either the user will have to live with this software loaded with excellent and cool features – most of which will not be used anyway – or the management will have to 'upgrade' the target configuration; which adds to the overall cost of the project.

To circumvent this problem make sure that the testing machines are the exact same configuration as that of the users and a module should fail in testing if it cannot meet the expected performance on these test machines. As user champions are also involved in testing, they can visualize and experience, first hand, the kind of performance they will get from the system once it is deployed on their desktops.

This assumes that the performance and other development guidelines are set well before the project starts and have been signed off by the users.

Incorrect Prioritization

This factor comes into play when things start going wrong or the deadlines start slipping. As the pressure mounts on the team to deliver on time, the project manager and the team start juggling priorities and start being creative about how best they can meet the deadlines.

More often than not, the first task that gets slashed is the duration of testing, which is the biggest mistake a team can make. Unless the system is tested to the satisfaction of testing team we may never know how it will fare when deployed on users' desktops. Testing also provides good feedback to the managers and team leaders about the strengths and weaknesses of their team members and the team as a whole. This information becomes crucial in order to manage the project and allocate appropriate resources for each task.

Whenever the need arises to re-prioritize you will have to decide between tasks that are important or urgent. Always focus on what is important for the project even though the gains may be intangible rather than something that can be measured and computed or shown on a graph.

Underestimating the 'Human' Factor

This factor does not, to a great extent, impact the deadlines or budget of a project but is a silent killer for the overall quality of delivered software. Working with teams means working with people and no matter what management principles or fads we use, ultimately it comes down to how well you manage an individual.

This means knowing your team members better – about their strengths and weaknesses, motivations and aspirations, etc. Your task allocation should be based upon the strengths of the individual team member rather than availability. Yes, no one wants to be in the testing role, but an individual having attention to details will provide much better feedback from testing than the one who is a visionary and works best with ideas.

The intangible benefit of assigning people the tasks that they are comfortable with and complements their strengths is that you not only have high quality results but also have a team that is motivated to work together. This, however, does not mean that just because a person is detailed oriented s/he should be stuck into a testing role forever. A balancing act will be needed to diversify the skills and let each person in the team learn from others, which in the end will be beneficial to the individual as well as the team.

In summary, I would like to say that even though a lot of factors influence the success of a project, there are very few that impact it in a big way. The factors I have listed have consistently appeared on the radar screen of almost all of the projects I have been involved with and am confident that a lot of you will be struggling with at least couple of them.

Unfortunately, there is no magic formula that can be applied which can guarantee the success of a project but it sure helps to side step or be aware of the factors that can almost guarantee the failure.