Grow Up Dude! (vol #1)

Raghavendra K S (rags_1979@yahoo.com)

12th December 2002

"Grow Up Dude!" articles are based on some personal experiences and talk about various random aspects that can be bettered in a software services company.

I am a software developer working for a reputed organization (with around 300 employees) in Bangalore. We are a Services company. What I present in here is the second in the series of articles under "Grow Up Dude!" and contains my personal views and the company I work for has no explicit endorsement for any of the views mentioned in this article.

Everyone wants money! To get business running an organization goes ahead and commits really challenging propositions in terms of effort and time lines. Then they inevitably slip the committed deadlines.

There are three ways to solve this problem.

1. Given an organization's work "efficiency", we learn to estimate correctly.
2. We stick to our estimates and improve our "efficiency".
3. Do not over-commit to the customer. We should better our customer expectancy management.

The best solution should be worked in all three ways. That is to say that we need to know the ground reality while estimating and yet, at the same time strive to improve on our performance levels.

Accuracy in estimates can improve… if only people want to!

This is one of the toughest things in the software industry. How do we know how much time a particular task gets done before it actually is done? Theoretically predicting this with a 100% probability is impossible. While we understand that estimates can never be perfect, we should realize that the error made should be as minimal as possible.

How do we make sure that our estimates are as precise as possible? One of the answers is being honest when we estimate. It is great to feel optimistic and confident about one's team, but optimism and confidence alone can never see a project to its successful end. There are various factors, which include relevant skills, availability of tools, domain knowledge, etc on the technical front and interpersonal relationships, leadership etc on the softer side. Confidence in the team's and one's own abilities is a must for success of any project. But there should be a check- " When I say, I know this stuff…. Do I really know in and out of it? Am I excellent at it?"

Another important point that should be kept in mind is that software is getting complex day by day. What makes software so special that it can

escape the famous third law of thermodynamics? The entropy of our systems is just shooting up day by day. This is increasingly true especially in the application development areas. So it is best to leave the estimates to the developers than managers doing it. Because the way software was built in the days when the manager was a developer is lot more different and complicated in today's world and the developer is the best person who understands it to the core. Pay heed to his pleas and cries! The developer may lack the experience to estimate in which case the alternative is not the manager estimating. Rather, the developer should be given some help and guidance to come up with the numbers. This way, there is an added advantage of training the developer also.

There is another common pitfall when a manager estimates. He could be a technical expert or knows very little about technology. Either case can sometimes be dangerous when it comes to estimates. He tends to estimate the time it would take for him to complete the task and assumes all developers would take the same time. I want to call this as "I have seen it all" syndrome. It might take some task 5 days for him because he is an expert, but then he is not the one who is going to develop it. Probably that module is going to be developed by a slow learner or someone who never had exposure to the particular domain. So probably it would take this guy 10 days to develop. We should actually consider the abilities and domain knowledge of people when we do estimates. May be, it contradicts the idea of process centric organizations, where processes than people matter more, but then reality is that people drive the show and not the processes. People talk about processes and stuff like Six Sigma, people have been searching for a substitute to human capabilities in processes. Only time will tell if that is ever feasible and whether heroic performances are not really needed with just a well-defined process taking care of everything! And it is a different story if the manager (who is estimating) is not very comfortable with the technology. He has no clue what hidden intricacies lie in the system. To him, the module to be worked on might seem so misleadingly simple and then again estimate 5 days for a task that takes 10 days!

When venturing into newer technologies, nothing ever should be underestimated. It could seem very simple at first blush. But you never know what is in store for you when you embark on these technologies. I would like to quote Linus Torvalds on high-level design here. "But that's like saying that you know that you're going to build a car with four wheels and headlights -it's true, but the real bitch is in the details."

A personal experience has been with use of tomcat and xml in one of the projects. Nobody ever anticipated the problems or learning involved in using tomcat. When questioned, people just smirked saying "it is just a web-server!" But the fact is that we spent more than 10 man-days to

figure out exactly how things work on tomcat and how to use ant and other stuff. Similar experiences were felt when we tried to leverage xml/xsl technologies. We went ahead and didn't budget for any glitches. Painfully, it was realized after implementation that there is a problem when an xml from a remote web-server tries to access a local xsl sheet for displaying. Did we dream of this? No! But that is no excuse. It would be a huge surprise if someone told me that projects just run without any technical issues like these. We need to budget for these unknown issues. One of the metrics to qualify a good design is that it is most resilient to changes. The same is the case with planning. A good plan is one that accommodates all these unforeseen changes (at least most of them). In other words, we need to see a line in our project schedules saying *"I don't know what this is for…. But I allot these 10 more days because my experience shows that we will hit on some unforeseen glitch..."*

A best way to learn to estimate is a very simple technique. Next time before we start our work let us make a note of the time we thought it would take. At the completion go back and compare the actual effort with the estimates. It will never surprise me that most people most of the times find that they actually took longer time to complete than what they estimated. However, they don't realize this because they fudge the numbers or convince themselves saying, "It took those ten extra days because that module conked out unexpectedly, which it shouldn't have. So my estimates are correct!" Well, I would be curious to know one single project where everything just went fine with no unexpected hassles! By fudging those numbers one might feel comfortable about his abilities to estimate, but the harm is that he would repeat the costly error again and again. If one is honest, we will not find that repeated allotting of 6 days for a work that actually takes 21 days to complete!!

One of my colleagues always says, "If you need to fail, fail fast". In fact, methodologies like XP have grown precisely for this. However, it has been observed a number of times that we want to hide from the reality. A lot of us don't want to move from the comfort zone. We hate to hear that something takes more time than what we estimated it to take. As a result, people are discouraged to express their fears in advance about the schedules. The reader is encouraged to read Mythical Man Months where these things are illustrated very beautifully. But being realistic is as important as being optimistic. We need to be confident about our capabilities, but for some reason, if we think that something takes more than what it is estimated to take, we need to accept it. Because, failing at a later stage (at release time or Integration time) is often more expensive (the team size is the maximum at that time, the customer might have made arrangements to receive the software which is not ready in reality, etc)

Before moving on to the next section, I want to remark that

effective management and allocation of resources is again a very crucial thing. Splitting of a developer time on too many activities actually never works well. At management level, this time-sharing is fine but a developer needs to really get into nuts and bolts of the system and so should always handle just one thing at a time or at most two.

You need to Step Up!

Vision is the most important thing that the team leaders need to have. They need to make a good mix of achieving short-term goals as well as long-term goals. Project is very important but one should realize that the lifecycle of an organization is not restricted to any one project. The vision needs to be shared with the team members and team members striving to achieve long-term ambitions should be given good incentives.

Almost inevitably longer-term ambitions always clash with the present day needs. One should select some of these and be ready to sacrifice a little today for a better tomorrow. The best example I can think of is building generic and reusable components. In most of the projects, there is very low incentive for the pains the developers take to make this happen. This is because there are no immediate sparkling results. The results would be seen probably a year down the lane when the component which was built as a generic one today, was seamlessly customized for another project down the lane and effectively cut down 3 man months of effort. But probably, by that time, the developer is working somewhere else to receive recognition for his work. A developer sees no motivation to develop such components. In stead, every one tries to be an instant hero by making the system just work for today! So, time should be spent in evaluating and encouraging such a culture of building something more than “school-boy” stuff.

I want to make a comment on reuse. Reuse doesn’t essentially mean code reuse only. It could be design reuse or even concepts reuse. The advantage of reusing designs and concepts is that we are confident that they work and are stable because they have been tested earlier.

Reuse and automation in my opinion are the only ways we can survive the mad rat race called competition.

Now that we talked about reuse culture, let us get into another very important catalyst in helping us perform better – automation!

How many days would a code-review happen? In one of the very early projects I worked in, the code-review process was allotted around 20 man-days. Do we really have such a luxury? There are a plethora of tools that can do this for you. Invest time in exploring some of these once and you can use them every time. They will be more accurate than a human. This in no way is a substitute for experts’ involvement in code review. Human involvement is still needed to improve performance of the code or

to catch some bugs in the code but the tools can be the first-level filters. What takes 5 people 4 days to do first level filtering (checking for compliance of coding standards) can be done exactly by one person in 10 minutes. See the gains? It is an initial investment for someone to explore (it took me 2 days to explore this), but then it saves colossal amounts of time in the longer run – time and again! Not to mention how boring something like code review just for code compliance can get to when done manually!

There are tools for automating almost any mundane task you can think of. They are out there for free. There is checkstyle for code review, Importscrubber and Jindent for taking care of extra import statements in java and proper indenting in java. There is CruiseControl for doing automated builds. There is ant for compiling java programs. There is junit, xmlunit, dbunit etc for doing automated testing. There is Jdepend for finding out the quality of your design and NoUnit for finding out the quality of your test cases. Just think of anything and you almost have a tool for them.

Just to summarize, automation is especially useful when we know that something needs to be done again and again. It is less error prone than a human doing it and is less boring and a lot times faster. We shouldn't be tempted and compare that automation of something takes 5 days, whereas manual execution takes 2 days. Think of it, if you need to repeat the task 4 times, you have already saved 3 days!! And better still, it is more accurate, consistent and repeatable!

Along with the use of these tools, we should be open to new methodologies like Agile methodologies. They are radical, but probably we can pick some good things from them. Test first programming and automated testing are the best take-aways for us, to say the least.

We need to be open for changes and really experiment. Generally, any book on personality development would advice that "comfort-zone" is most often detrimental to your growth than useful. There are tons of books on this. But my favorite is "Who moved my Cheese".

There is another factor that can give us a competitive edge. That is to keep us technically abreast with the latest developments and strive towards constant innovation. People in the organization need to be encouraged to participate in seminars and developer forums. This enthusiasm is glaringly missing in us. How many of us really know of (say) Agile Methodologies? How many of us here have really used design patterns in our projects? While we should be cautious not to introduce too many new technologies all at once into a project because it makes it very risky, we should always attempt to embrace and familiarize (and master) newer technologies.

Technical depth is a must for all developers and excellence to the last mile should be encouraged. The

common tendency is to encourage developers learn things that are barely needed to make the system in hand working. In fact, surprising may be, but developers are sometimes discouraged to go deeper into the issues. I have seen that a lot of developers have developed an attitude to know "just what is sufficient for present". While it important to "get the damned thing working today", design quality cannot take a backseat for lack of time.

My experience with development shows that this superfluous depth is sufficient during smooth development, but when you actually are debugging you are simply lost! There just is no substitute for technical excellence. In fact, a strong technical foundation is necessary for everyone in the team. I have had the opportunity to interact with Martin Fowler and posed a question to him. "Martin, every methodology tries to address something or the other… Did you ever think of a methodology that can take care of in-competency?" He just was wild when I asked him that… and to quote him verbatim "If some methodology claims that it can take care of in-competency, then they are just Bull Shitting"

A technically sound design saves huge amounts of efforts in the later stages of the project. To ensure that we have strong designs, reviews should happen at appropriate times and help give team a direction. Reviews should be of highest quality. Every review should be a learning exercise to the people involved. If a reviewer says, "Everything looks fine" then I fail to understand his existence! The reviewer should also be shown "stick and carrot" approach depending on the quality of review he performs just like other developers are treated.

All developers need to actively participate in open forums and keep themselves updated. One of the reasons that this doesn't always happen is probably because no direct incentive is perceived for doing something like that. Finishing projects on time is important, but how do you finish them when you are equipped enough? It is important all developers realize the importance of being up to date and management realize that to do complete a task in time, their army needs the latest arsenal and training.

It should also be kept in mind that the developer in an organization is not restricted to any one project. The depth he has gained in a project (even if not used in there) will be useful somewhere else in the next projects to come. We always need to remember that we work for an organization and not for a project.

Developer communication is another area that is very important. Alistair Cockburn had done some extensive research on this aspect and the reader is encouraged to read some of his articles from his website (http://members.aol.com/acockburn). Even speaking from a personal experience, one of the projects started off with people sitting in different floors. There was a good suggestion and everyone was seated close to each other on the

same floor. It became so much easier to work. It definitely makes a lot of difference. We should insist that the team working on a project be closely seated wherever possible.

Conclusion

For success, management of an organization needs to be honest in their estimates and be open to face the reality. They need to believe that projects can run on schedule if planned properly. While they can improve on their estimating abilities, they also need to gear up all employees for better performance levels by earnestly trying to work towards reuse, automation and gaining technical depth. If there is need to change, then one shouldn't be scared to do so. Senior people should push these things forward because they have more maturity and a bigger picture of the organizational needs than the youngsters.