About making software projects

The standard request

Quite often we get prospective customers calling to consult us regarding making a software project. Typically they have some business objectives that can’t be achieved by purchasing some canned software because one or more of several possible reasons:

* Off-the-shelf software can’t meet their needs
* They need to be able to change, and canned solutions are quite rigid.
* Off-the-self software may not be compatible with other software they already have.
* Their competitive advantage relies in how they use the technology in a different way than their competitors

A lot of them come with a list of requested functionalities they believe they need in order to achieve their business objectives. Then they put to us the two following questions: How much is this going to cost? and how long is it going to take?

The standard process

This is a quite standard way of thinking and in the software industry there is a standard process to deal with the questions of schedule and cost, that consist in elaborating, double checking and understanding the list of things to be done, and breaking then down in tasks and subtasks until the necessary level of detail. Then a qualified member of the technical team should estimate or make a prediction of the necessary amounts of time, qualified persons and tools requested by every one of them. They should also look after the sequential order of the tasks, which ones if any can be executed in parallel, and the mutual dependencies among them.

The result of that should be a plan, that you can typically represent in a GANTT chart, with an associated estimation of the timing for starting and finishing and for the project cost. This process is called software estimation.

Issues with the standard process

It all sounds logical, intuitive and common sense, however there is still an unresolved issue with the traditional approach and this is the high variance that has been observed in practice between software estimations and real project execution. Steve Mc Connell explains this very well in his book: ‘Software Estimation: Demystifying the Black Art’. When he applies the Cone of Uncertainty Concept to a number of software projects the variance obtained ranges from 0.25 to 4, which means an uncertainty factor of 1 to 16 times.

This creates a challenge because in business terms to say to your internal or external customer that their project can take an unpredictable number of months between 1 and 16, doesn’t look to be very helpful.

Since the book of Steve Mc Connell was published in 2006 a lot of good work and research have been done on the art of software estimations, and you could expect some improvements in managing the uncertainty, however when you look at more recent statistics you still see similarly disappointing results. Just to mention a couple of them:

# McKinsey&Company has published in October 2012 in their article: ‘Delivering large-scale IT projects on time, on budget, and on value’ the results of a research conducted together with Oxford University over more than 5400 IT projects, and they found out that 66% of all the software projects ran over budget, 33% ran over schedule and on top of that on average they delivered 17% less value than predicted. <http://www.mckinsey.com/insights/business_technology/delivering_large-scale_it_projects_on_time_on_budget_and_on_value>

The Standish Group is publishing every year a snapshot of the state of the software development industry. This year they have studied 50.000 software projects around the world and found out: only 29% of all projects have been completed successfully; 52% were overrun either on time, on cost of both, and 19% were failed. <http://www.infoq.com/articles/standish-chaos-2015>

Reasons why the standard process doesn’t produce accurate estimates:

In his 2006 article: ‘Schedule Estimation and Uncertainty Surrounding the Cone of Uncertainty’ Todd Little identifies the following reasons for estimation variance:

* optimistic assumptions about resource availability
* unanticipated requirements changes brought on by new market information
* underestimation of cross-product integration and dependency delays
* a corporate culture using targets as estimates
* customer satisfaction prioritized over arbitrarily meeting a deadline

<http://www.toddlittleweb.com/Papers/Little%20Cone%20of%20Uncertainty.pdf>

All of them except the forth one have to do with basing the estimation on assumptions made at the beginning of the project. But they are later challenged by the reality; thus resulting in the estimation to be wrong and failing to predict the schedule, the cost and even the scope.

Even in a theoretical case where we would consider all of the previous assumptions to be accurate still there is an additional human factor, consisting on an optimistic bias when we predict the time we will need to complete a complex task. This phenomena called ‘The Planning Fallacy’ has been studied by Nobel Prize Daniel Kahneman and explained in his best seller book: ‘Thinking Fast and Slow’. The optimistic bias has been observed to happen regardless of what our previous experience is in dealing with tasks of a similar degree of complexity. <https://en.wikipedia.org/wiki/Planning_fallacy>

Taking a different approach:

In 2001 some of the best software developers in the world came together to a 2 days long meeting to discuss about the problems in the software development industry and to find an approach that could help to avoid them. The result of this meeting was the **Manifesto for Agile Software Development** and the the **Twelve Principles of Agile Software**. <http://agilemanifesto.org/> and <http://agilemanifesto.org/principles.html>

The agile manifesto define a new set of values that are considered as a priority over the traditional ones, and a set of principles that facilitate the application of the values. This is the approach that we apply at ASPgems. Out of the several methodologies existing for the AGILE approach we have chosen the Scrum one.

The Agile Team and Collaboration

To work with this methodology we create multidisciplinary teams, including at least a member of every one of roles that are necessary for the project. As we collaborate with the customers, they participate in the team as well. We divide the work into small chunks called sprints, and at the end of every sprint we deliver a chunk of working software and we sit together with the customer to review it and to plan for the next Sprint. We take in consideration all the new feedback and any new events during every sprint thus allowing us to adapt and respond to changes.

By doing this we collaborate with the customer, and by including in the team all the development roles and all the stakeholders from the customer we make sure that we adapt the software to the evolving needs of the customer.

Working the Agile way

To do this we start by understanding the customer’s context and defining clearly what the customer’s business objectives are and what their key factors of value are.

Then instead of asking ourselves the question that we mentioned at the beginning of this post, we propose to change the question: rather than coming with a fixed scope and request how many resources and how much time are requested, we propose to change the question and ask to ourselves and themselves what is the best functionality that can be provided to meet the business objectives and the Key Factors of Value within the available timeframe, and with the available budget.

In the classical approach we try all and everything to control the situations in order to force if necessary to meet the plan. While in the Agile approach we try to adapt to the situations. In the classical approach we focus on meeting the plan and the list of functionalities, and the timing, while in the Agile approach we focus on what provides more value to the customer.

The project with the Highest possible Customer Value.

Then we create a low granularity scope, which means a list of necessary modules and functionalities the software will need to have in order to deliver the necessary value, but without descending to the level of detail.

Once we start the development process it is at every sprint where we make the transition from low granularity design to high granularity, and we do it by making decisions together with the customer, while taking in mind the key value factors, and the constraints.

By this adaptive and collaborative process we make sure that the customer’s business objectives are met, and on achieving this we are guided by the key value factors. We avoid to invest a lot of effort into defining things with high fidelity before we can make sure that we won’t need to change them. We manage to get all the customer’s knowledge transferred to the project team; which is not possible if you pretended to concentrate all of this transfer of knowledge only in the initial analysis phase. We also make sure that the experience of the users is best satisfied as we are learning from their behavior during the process.

By doing all of this we make sure that the software will deliver the maximum possible value within the schedule and the budget constraints that have been given.

Cambio n1 para probar GIT