Agile Fixed Price Projects part 1: "The Price Is Right"

Pascal Van Cauwenberghe
Nayima
pvc at nayima.be

Introduction

When I talk about Agile Software Development methods [Highsmith 2002] and Extreme Programming [Beck 1999], the remark I get most often is: "***That will never work!***" The only reply I can give is: "*It works for me*". The second-most often heard remark is "***I can't do that, I have to do fixed-price contracts!***" My reply to that is a bit more involved. This two-part article contains my reply, so that maybe next time I will get some different remarks.

This article describes an approach to fixed-price projects using a classical, "rigorous" process. There are three phases in the process: **qualifying**, **selling** and **implementing** the project. This text provides some tips for each of the phases:

- First of all, I have to decide if a project can reasonably be done under a fixed-price contract. This text contains some questions I ask myself to (dis)qualify projects.
- If I decide to go for the project, I still need to win the project. The sales process is crucial to set up the right conditions for implementing the project. The sales tips allow me to remove some major implementation risks before the project starts.
- When I've won a contract with the right conditions, I can implement and deliver the project, using the implementation tips in this text.

All of these techniques have been applied on succesful fixed-price projects. How do I know they were successful? Because, given the choice, everyone involved would do the next project the same way.

A fixed price contract

A fixed-price (FP) contract between a **provider** and a **customer** defines the scope, features, planning, timing and price of a software project. Pretty much everything is fixed, not only the price.

Why do customers so often demand "**FP**" contracts?

- If projects are awarded after a multi-provider bidding process, the customer needs to know scope, timing and price to choose between the bids.
- Customers think that they take no functional risk: if the provider does not deliver on time or on spec, the customer can always sue! Of course, if the customer really needs the software on the given date, they too are in trouble.
- Customers think they take no financial risk as the price is known beforehand. However, surprisingly many "fixed price" projects cost more than initially agreed upon. We'll see later how this can happen.
- The defined, planned and sequential project flow gives the customers a warm, safe feeling of control. Until near the end of the project, when these projects so often "suddenly" start to fail.

This type of contract is almost universally hated and feared by software providers because of their high financial and functional risk and their low success rates. The contract seems to protect the customer at the expense of the provider.

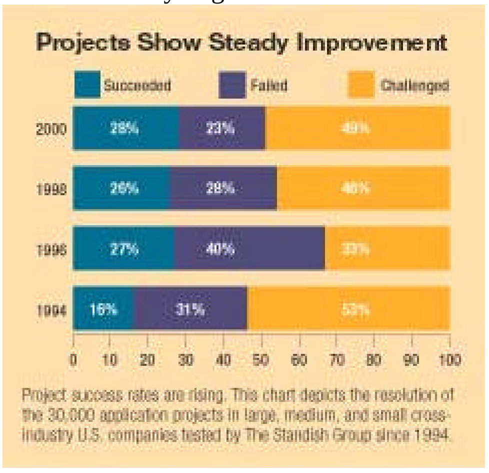
Do you have what it takes?

What does it take to be successful project manager of such a project? I and my team will enter into an FP contract if

> We can fully specify, estimate and plan the project.
>
> We can deliver the product exactly as agreed, within some small tolerance.

Do you fail one or both of these criteria? Why would you enter into an FP contract? **You will very likely lose money and your customer.**

You are not alone: according to the "Standish Group Chaos report" [Johnson 2001], 72% of projects fail to deliver what was originally specified, in the agreed time and on budget. Read this text to get some tips to learn how to evaluate and satisfy the criteria. And don't do FP contracts until you get the risk under control!



The following questions refine the conditions, which are necessary for me to make these statements.

Question 1: Do I know the domain?

I can’t specify, estimate and produce a software product well enough if it’s in some domain I’ve never worked in before. That would be too much to ask.

I need to know the domain well to be able to **specify** the project: I need to understand the language and problems of the customer; I need to know where to get more information; I must be able to devise solutions and explain them to the customer. I must be able to help the customer to draw up a complete specification by asking questions like: “Do you need feature X? The customers in the four previous projects in your domain all required X”.

I need to have experience in the domain to **estimate** correctly enough: I use actual effort measurements of earlier similar projects; I must be able to distinguish between what’s comparable and different between this and previous projects; I must have experience with the common risks of the domain.

I need to know the domain to **execute** the project well: most of the project should be the application of known techniques, with as few as possible original problems and solutions; I expect to be able to foresee and handle most of the risks in the project.

Question 2: Do I know the technology?

I need to be familiar with the execution environment (computer and operating systems), the programming language, the tools and the development process. I don’t experiment or use “bleeding edge” technology on FP projects, only boring bread-and-butter methods and tools that I know and master perfectly.

Question 3: Can I work with my usual team?

The most important variable is the development team. What’s the use in knowing that last time we implemented some feature in 20 days with team X, if team Y implements this project? How quickly and how well will team Y implement this same feature? I have no idea, unless I’ve worked with them on similar projects.

Team performance depends for a part on the talent and experience of its members. Performance depends a lot more on how well these people work together. If a team works well together, has a “rhythm” and knows its “pace”, you can predict with some accuracy how they will perform. If the “team” is a bunch of people freshly assembled for the project, expect to lose time building a team and forget about estimating their performance accurately.

Add newcomers to a team one by one, to teams of experienced people.

Don’t count on them to add much to the project, at first.

Question 4: Can I handle the estimated project size?

Am I comfortable with the length of the project? If I’m used to doing projects shorter than 6 months, I will be taking a huge risk when I take on projects of 1 year or longer. If the project is longer

- There are more requirements to analyze, estimate and plan.
- Planning and estimation errors become larger, as the estimations of the later parts of the project are based on the correctness of those of the earlier parts
- Changes in requirements become more likely and more necessary as the environment of the system changes.

Am I comfortable with the number of people on the project? Managing a team of 50 is fundamentally different from managing a team of 5. If the team is larger

- Communication overhead becomes larger (relative to the square of the number of people)
- Misunderstandings and miscommunication become more frequent
- More effort has to be spent dividing and synchronizing the work
- It becomes harder to find shared vision and values
- Change becomes harder as we have to convince more people

Bigger projects require more time and/or more people. Longer projects and larger teams are less efficient than shorter projects and smaller teams, because the effects of size are not linear.

Sales tip 1: Don’t just respond to RFPs

Customers often look for a provider for a fixed-price project by sending out a “**Request For Proposal**” document. The RFP contains a description of a problem to be solved. Providers who wish to implement a solution, have to respond with a written proposal containing a specification, timing, planning and price. The customer then chooses the provider with the best proposal, according to their own criteria.

This customer has, most likely, been helped to write this document by one of your competitors.

As a result, RFPs, which should be open-ended, typically have a concrete solution in mind: your competitor's solution. And, in any case, these RFPs are always incomplete.

Just responding with a proposal document is not very likely to win you the deal. Even worse: you might win the deal, but if you base your specification, planning and estimate upon this biased and incomplete RFP, your project will most likely fail.

I go and talk to the customer, ask questions, get some more information, get the answers that are not in the document, establish a rapport, try to steer them away from the solutions already envisioned with my competitor during the drafting of the RFP.

If they won't talk to me, answer my questions or clarify their wishes now, I don't make a proposal. I don't have enough information to work on. Even if I could win the project, how likely is it that I will communicate better during the project?

Sales tip 2: It works both ways

A fixed-price contract is a contract between two parties for their **mutual benefit**. Both parties have rights and responsibilities and these must be divided fairly between the two parties. If either of the parties does not feel treated fairly, I don't enter into the contract. The contract should clearly state the responsibilities of both parties. E.g. the customer should deliver some information by a certain date, provide testing feedback within a certain timeframe… The provider should deliver some functionality by a certain date, make the product comply with certain quality criteria….

More important than the contract is the working relationship of the customer and the provider:

- Is there a good level of communication?
- Do both parties trust each other?
- Are both parties willing to perform their part of the job?
- Does everyone realize the commitment they are making? Do both parties have the necessary time, knowledge and authority to do their job well?
- Is there a willingness to solve the problems that will inevitably arise?
- Is everyone committed to making a success of this project?

One of the most important tasks during the sales process is to set up this working relationship. If you fail to do that, you've just added a huge risk to your project.

Sales tip 3: Don't underbid

I can estimate a project perfectly (for some definition of "perfect"): I know how long the project will take, how much it will cost. Thus, I can compute a price that allows me to recoup my costs and make some decent profit.

If you're in competition to get the project, it will be tempting to lower your price, planning to go over budget anyway. This extra billing might compensate for the loss you make on the initial bid.

I don't enter into a contract that is unfair to me and I don't try to correct this unfairness by not giving the customer what was agreed. That would a great way to start a business relationship…

Do I tell my team that their target is not feasible? Do I tell them that I expect them to fail? That would be a great way to motivate them…

And it doesn't work anyway, because "Implementation tip 1: Don't allow change requests" doesn't allow me to increase the budget. It's a fixed price contract, remember?

Sales tip 4: Add some slack to cover the risks

I have to admit it: **I can't specify, estimate, plan and execute a project perfectly**. It's a useful and reassuring simplification, but I'm hardly perfect. For projects in a known domain and environment, with a known team and of the usual size I can get close. But I know I will make mistakes before and during the project.

There are factors related to the customer that can't be controlled: how well will they respect their commitments, how well have they specified what they needed, how high are the odds that the requirements will have to change…?

Then there are the "forces of nature" that I have no control over: people will get sick, computers will throw tantrums, and other jobs will need to be done urgently….

All of these foreseen events and many more unforeseen ones are the "risks" of the project. I try to enumerate and quantify the risks, the odds of their happening, and the cost of avoiding or mitigating them. And then I add some more for unforeseen risks. I add a few percent "slack" to the estimates (and thus the planning, timing and budget) to cope with all these risks.

How much slack do you need to add? I've added between 10% (for predictable, short projects for known, professional customers) and 30% of the original estimates. If I feel I need more slack than that, this project is probably too risky to do under a standard fixed-price contract.

I don't cut slack to underbid a competitor, because I will need it during the project, if I'm paid for it or not.

Sales tip 5: Real business requirements

I write the specification together with the customer. If they don't have enough time to discuss, review and improve the specification, I don't bid for the project: if the project is not important enough to specify and plan well, it's not important enough to implement it.

Each item of the specification, each feature (or use case or user story) must comply with the following criteria:

- The description of the feature must be fully understood by the customer and by the development team. The description uses a vocabulary that is familiar to the customer, no technical mumbo-jumbo!
- The feature must add some business value. The customers must understand why this feature is included, what value it will provide.
- The feature must be verifiable by the customer. At some point I have to ask the customer "Is this requirement met? Yes or no?" If we've defined the acceptance criteria beforehand, I can be confident that I will get a "Yes".

I leave the technical details out of the customer's specification; they're only used within the development team. The specification should be as brief as possible, so that we don't need to spend an inordinate amount of work writing it. The specification should describe the requirements with "enough" precision: just enough to be able to understand, estimate, implement the requirement and to evaluate if the implementation meets the requirement.

Implementation tip 1: Don't allow change requests

Change Requests are a well-known tool used by most project managers. If a customer wants something that's not in the original specification, their project manager can fill in a "Change Request" form, which describes the change. Based on this information, the provider's project manager can estimate the extra work and cost required. If the customer agrees with the estimate, the extra work is performed. The cost of this work is added to the final bill, the extra time is added to the planning.

Change requests have the following **advantages**:

- They allow the customer to steer the scope of the project, use the knowledge they have gained during the project and correct any mistakes made during the initial specification phase.
- The provider gets to bill more than budgeted, which makes the provider's CEO and CFO happy.

But there are many **drawbacks**:

- The changes disrupt the orderly flow of the project, making the development team less efficient. Team members get demoralized when feature lists and planning are in a state of flux and completion dates slip.
- It's more difficult to schedule and synchronize with other projects, as there's no way to predict when this job will be finished.
- The disputes preceding the change request ("It's in the spec! No, it's not! Yes, it is…") and the haggling over estimates and extra cost poison the relation of the provider with the customer. All of this nasty commercial negotiation stuff should have been finished before the project started.
- Change requests invariably lead to dissatisfaction of the customer as the budget and timing creep. How does a project get to be late and over budget? One change request at a time. Welcome to the "challenged projects" category!

The problem with change requests is that their negative effects only show up after a delay. Responsibility for the project at the customer is typically shared between a project manager (with authority over functional matters) and the finance manager (with authority over budgets). The project manager agrees with every small change request and is happy to see more functionality added. The finance manager only sees a large budget overrun at the end of the billing period. The end users only see that the product is not delivered on time. By the time the negative effects appear, it's too late to do anything about it. And so, a lot of yelling and recriminations ensue… Which makes everybody unhappy.

Don't use Change Requests because their drawbacks heavily outweigh any initial advantages they bring.

When I explain this rule, older, wiser, more experienced and more cynical people invariably point out to me: "*You'll never get rich this way!*" Change Requests seem to be a standard technique to make customers pay more than agreed.

I'm not rich, so I guess they are right. But is a project manager who brings in twice the amount budgeted, by delivering late, a success or a failure? The use of "Exchange Requests", explained in the follow-up article, can deliver the flexibility of change requests, without the negative effects upon schedule and budget.

Implementation tip 2: Spend your slack wisely

Thanks to “Sales tip 4: Add some slack to cover the risk”, the project has been allotted more days than strictly necessary. I use this extra time sparingly. I try to resist the temptation to “slack off” because there’s ample time left. When do I spend slack?

- Something goes wrong: a team member get sick, a tool doesn’t work as advertised, the database crashes, the computers refuse to work, some risk materializes… Accidents happen, some of the slack must be spent to handle the problem.
- An estimate is too low. If a job takes 2 days more than estimated, I take 2 days out of the “slack piggy bank”.
- The specification was wrong or can be improved. I spend a small amount of slack to do the extra work required, so that we can avoid “change requests” or other actions that increase timing and budget. The customer appreciates it if I spend my precious time on improving his system.
- Something unforeseen happens. I can try to foresee and avoid all possible risks and still some unforeseen ones crop up. I’m constantly on the lookout for these events and use some of my slack time to nip them in the bud.

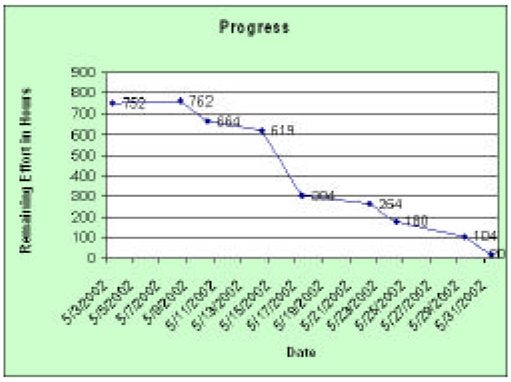
Sometimes things turn out better than expected, jobs get finished faster than estimated. Put the time you gained back into your “slack piggy bank”.

Imagine a 19th century engineer keeping a steam engine running. The project is like a beautiful machine, lovingly built, gleaming clean and in perfect working order; slack is like a small can of lubricating oil. A few drops of oil here and there do wonders to keep the machine running smoothly. At his best, the engineer is completely attuned to the machine and seems almost prescient, lubricating the parts before they start creaking. On the other hand, no amount of lubrication is going to keep a badly maintained, sloppily built and over-stressed machine doing useful work.

Implementation tip 3: Simple, honest and correct tracking

During the course of the project, I need to track my team’s progress. Are we behind or ahead of schedule? Will we be able to deliver as promised? Do we need to take some corrective action? It all sounds very complicated and time-consuming. There are all of these wonderful, expensive and complicated tools I can use. Do I really spend a lot of time tracking? Of course not. The team and the customer only need to know two things:

- Will we be able to deliver as promised?
- If not, what can we do to get back on track?



A simple and effective method is to have a “burndown chart” or “backlog chart” [Schwaber 2002]. This essentially plots the amount of effort left versus time. Each time a feature is finished, we reduce the “amount of effort left” by the effort estimated for that feature. Any child can see how we’re doing. This chart is easily updated and should be visible to all project participants, as an “information radiator” [Cockburn 2002]. If anyone wants to know “Are we there yet?” they just have to look at the chart. It’s important to only count fully completed, tested and “ready for acceptance” features. This keeps me from deluding myself with statements like “the feature is 80% finished”. A feature is either done (and acceptable for the customer) or not done. This guarantees that the tracking represents real progress.

The plan is just a plan; the only important thing is the delivery of the project. I don’t care about deviations, as long as the goal of delivering is not jeopardized. For example: if I have two features, each estimated at 5 days, I don’t worry if one takes 4 days and the other takes 6 days. Sure, I didn’t follow the plan, but I’m still on target to deliver as planned. The plan gets modified to reflect reality, but always with the same goal: to deliver the project as promised.

We can also record how long each feature took to implement. This allows us to calibrate the team’s speed and to improve the estimates for the following project.

Implementation tip 4: Manage your project

If the requirements have been established, the effort has been estimated and the project has been planned the hardest part is over, right? The rest is just implementation: following the plan. No. A project manager’s job is to **manage** the project. What does that entail?

- I'm aware of a lot of risks that the team runs. I've prepared ways to avoid the risks, devised contingency plans and stored some amount of "slack" to deal with them.
- I'm constantly on the lookout for new, unforeseen problems. Whenever one rears it's ugly little head, we nip it in the bud.
- If the tracking shows we are getting into trouble, the team and the customer know it and we find and solve the problem.

The job is not about specifying, estimating and planning perfectly.

The job is to deliver what the customer needs.

If we extend the rugby metaphor of the "SCRUM" method [Schwaber 2002], the team's goal is to score the touchdown. We're always on the lookout for adversaries threatening our progress. Everything that gets in the way of the team gets tackled. We don't care who scores the touchdown, as long as the team scores. The project manager (or "scrum master") is responsible to get all obstacles out of the way.

Attack your risks or they will attack you

Some people think the project manager should shield the team from every outside influence that might impede their progress. For example: "developers should not talk to the customer, lest they get confused". On the contrary, involve the team in discussions with the customer about functionality and let them avoid and mitigate risks and solve problems. The job of the project manager is not to solve all problems and shield the team, but to ensure that the problems get solved.

Don't shield the team.

Help them to avoid and solve problems.

With all of these tasks, the job of a project manager looks quite hectic. It isn't, unless it's done badly. A good project manager is proactive and solves most problems before they become apparent; which still leaves enough problems to fill a full working day. If problems grow and fester, the job becomes a lot harder.

Good project managers don't seem to do a lot and lead pretty uneventful lives.

What kind of project is this?

If we review all the tips what are the recurring themes? I've done this type of project, in this type of environment, with my team a thousand times before. I can look back on several similar projects for experience, domain knowledge and actual performance measurements. I do my best to minimize the risks and to keep as many parameters as possible constant from one project to the other.

It all sounds very boring: "been there, done that!" It's not boring at all, because each customer, each problem and each project is in some way unique and will bring some unexpected events to tax my project management skills.

This type of project is what Jim Highsmith calls an "**Optimization**" project [Highsmith 2002]: performing a well-known activity as efficiently as possible by reducing the risks. This type of project works best with a "Rigorous Software Methodology": a method whose main method of dealing with risk is to reduce or eliminate it.

From the description, this looks like a relatively narrow category of projects. The further we deviate from the low risk ideal, the more dangerous a fixed-price contract becomes for the provider and the customer. The next article will describe some techniques that are more suited to "**Exploratory**" projects, where risks are higher. I use these techniques to extend the range of projects that can be handled with fixed-price contracts.

Conclusion

Fixed-price contracts fix scope, timing, planning and price of a software project. They represent a high risk for the software provider and for the customer, even though they seem to shield the customer from all risks.

These contracts can only safely be entered into for low-risk "Optimization"-type projects. We've seen a few selection criteria that allow me to (dis)qualify a project for implementation under a fixed-price contract. If the project qualifies, we can apply the sales and implementation tips, which allow us to reduce certain project risks.

If the project does not fit the Optimization model, it should not be executed under a fixed-price contract with the process described in this text. The next article describes some techniques to handle projects with higher risks under a fixed-price contract.

References

[Beck 1999] "Extreme Programming Explained", Kent Beck – Addison Wesley 1999

[Cockburn 2002] "Agile Software Development", Alistair Cockburn – Addison Wesley

[Highsmith 2002] "Agile Software Development Ecosystems", Jim Highsmith – Addison Wesley 2002

[Johnson 2002] "Collaborating on Project Success", Jim Johnson, Karen D. Boucher, Kyle Connors, and James Robinson – Software Magazine February/March 2001. Online at

http://www.softwaremag.com/archive/2001feb/CollaborativeMgt.html
[Schwaber 2002] "Agile Software Development with SCRUM", Ken Schwaber and Mike Beedle - Prentice Hall 2002