2012

Agile Meets Fixed Price Contract

Mike McCormick

MPCS, Inc.

1/9/2012

Agile Meets Fixed Price Contract

Contents

Fixed Price, Time and Scope

Fixed price contracts freeze all three magical factors at once – money, time and scope. Is the price and time a problem for agile teams? Well they shouldn't be. In fact, time boxing is common agile practice. Limiting money simply makes time boxing work better.

A real problem with fixed price contracts is the scope, which is fixed in terms of what should exactly be built instead of how much we should build.

Why are clients so obsessed with fixing the scope? We understand that they want to know how much they will pay (who does not want to know that) and when they will get it. The only thing they don't know is what exactly they want to get (even if they insist they do know).

The problem of fixing the scope has its roots in:

- Lack of trust between the contractors
- Lack of understanding about how software development processes work
- Misunderstanding what the scope means

Every fixed price contract has a companion document, the “Requirements Specification” or something similar. This document tries to reduce the risk of forgetting something important and tries to set a common understanding of what should be done to provide an illusion of predictability.

Key **wrong assumptions in fixing the scope** are:

- The more detail we include in the scope definition up front, the better we understand each other
- Well-defined scope will prevent changes
- A fixed scope is needed to better estimate price and time

Moving from Fixed Scope to Fixed Budget

Now, when we understand that the main conflict between application of agile mindset and a fixed price contract lies in the fixed scope, we can focus on converting the fixed scope into the fixed budget.

This means that instead of providing detailed specification of requirements we need to focus on catching as many of them as possible in the form of user stories. We need to build an initial product backlog which we will try to estimate using the story-points technique along with whole-team estimation techniques like planning poker.

First, we need to understand that a higher level of detail in software requirements specification means two completely different things for both sides of a contract. Software companies will usually focus on technical details while the other side is more business oriented. Specifications are usually created by a software company and are usually very protective. This is the place where the three key players come to play.

User stories as a way of expressing requirements are understandable for both clients and vendors. The understanding builds trust and a sense of common vision. User stories are quick to write and quick to destroy, especially written on an index card. They are also feature oriented, so they can provide a good view on the real scope of a project and we can compare them with each other in terms of size or effort.

Story points as a way of estimating stories are harder to explain to the client in the beginning because they are not a common way of expressing the effort. But in return they lower the risk of underestimating the scope. How does this happen? Story points in their nature are relative and focus either on the whole scope or on the groups of stories, while traditional estimation (usually done in man-hours) tries to analyze each end every feature in isolation.

Definition of done is another way of building trust and common understanding about the process and all the future plans for the project. It's usually the first time Clients see user stories and while they may like the way the stories are written, it may be not so obvious what it means to implement a story. Teams who discuss their definition of done with the client know better what the client's expectations are. They also estimate better. In addition, on the client side, the definition of done creates high level criteria for user story acceptance.

For example the definition of done for stories in the newly created web application may introduce such criteria as:

- Tested under IE 7/8, Firefox 14.x and Chrome
- Corresponding section in the "Users Guide" created

Later on the team can put some other, more internal elements of the definition of done, but at the stage of building a contract those general guidelines give an understanding between the Client and the team about what we all want to build within a defined scope.

Hopefully, having our definition of done in mind we will come up with some value defining our **scope budget in points**. And this value, not the stories that are behind it, is the first thing that should be fixed in the contract. This opens the door for change.

While we have the scope fixed (in terms of points) we still want to embrace the change the agile way. We have the tools (user stories and points) which we can use to compare one requirement with another. This allows us to exchange requirements along the way within a defined scope limit. And if we can stay within that limit, we can also stay within the fixed price and time.

Initial Estimation

The hardest part in preparing a fixed price contract is to define the price and schedule that will be fixed based on the well-defined scope. Let's look how can the team contribute to such a contract with initial project estimation done the agile way.

First, **educate**. Meet with your client and describe the way you're going to work. We need to tell what the stories are all about, how we are going to estimate them and what is the definition of done. Often we need to do that even earlier when preparing an offer for the client's Request for Proposal (RFP).

The next step is to **gather user stories**. This can be arranged as a time-boxed session taking no more than 1 or 2 days. This is long enough to find most of the stories forming the product vision without falling into feature creep. At this point it is also very important to discuss the definition of done for stories, iterations and releases with the client.

We need to know:

- The environment in which stories should be tested (like the number of browsers or mobile platforms, or operating systems)
- What kind of documentation is required
- Where should finished stories be deployed so that the client can take a look at them
- What should the client do (i.e. take part in demo session)

- How often do we meet and who participates
- etc.

This and probably many more factors specific to your project will affect the estimation and will set a common understanding about the expectations and quality on both sides. They will also make the estimation less optimistic as it often happens when only the technical aspects of story implementation are considered by the team.

Having discussed with the client a set of stories and a definition of done, we can now start the estimation. This is a quite well-known part of the process. The most important activity here is to engage as many future team members as possible so that the estimation is done collectively. Techniques like planning poker are known to lower the risk of underestimation because of some particular team member's point of view, especially if this team member is also the most experienced-one, which is usually the case when estimations are done be one person. It is also important that the stories are estimated by the people who will actually implement the system.

The Fibonacci-like scale (1, 2, 3, 5, 8, 13, 20, 40, and 100) comes in handy for estimating stories in points. Relative estimation starts with finding a set of easiest or smallest stories. They will get 1 or 2 points as a base level for further estimation.

In fact during the initial estimation it is often hard to estimate stories using the lowest values like 1 or 2. The point is, the higher the estimation, the less we know about the story. This is also why estimating in points is easier at this early stage, because it is far easier to tell that a story A is 2x as complicated as story B than to tell that story A will take 25 man-hours to get it Done (remember the definition of done?) and the story B will take 54 hours.

This works well even If we choose 3 or 5 point stories as the base level and if we do that, then it will be easier to break them down into smaller stories later during the development phase. Beware however the stories of 20, 40 or 100 points. This kind of estimation suggests that we know nothing about what is to be implemented, so it should be discussed with the client here and now in a little more detail instead of just happily putting it in the contract.

The result of the estimation is a **total number of story points** describing the initial scope for a product to be built. This is the number that should be fixed in terms of scope for the contract, not the particular stories themselves.

Fixing the Price and Time

Total number of points estimated based on the initial set of stories does not give us the price and time directly. So how do we get them? We need some magic numbers.

We should not be surprised by this as in fixed price contracts there have always been the magic numbers. All estimated values have always been such numbers. Right now we are changing the way we estimate but at the end we also need to predict the future based on the knowledge we have of the past. So what do we need to know? It's the **predicted team's velocity.**

Let’s assume we estimated our stories for a total of 300 points. We need to determine the velocity that our team may have based on:

- The team's velocity on earlier projects
- The number of activities we need to do in our iteration (see the definition of done)
- The ease of communication with a client (of course the ideal and hard to achieve goal is to have a client on site as a part of the team, but usually this is not what we have and

clients have different amounts of their time that they are willing to spend with our team) – the less time our clients have for us, the lower our velocity
- Just a feeling we have when looking at the scope (yes, good old' wishful thinking)

Let's say we have a team of 5 people -- 3 programmers, 1 tester and 1 team leader -- who will communicate with clients and stakeholders as well as any other people outside of the team (like a Scrum Master in a Scrum methodology).

We may now decide that our team will try to achieve a velocity of 20 story points per 2 week iteration. This is our main magic number in the contract. Many factors during the project may affect the velocity, however if the team we're working with is not new, and the project we're doing is not a great unknown for us, then this number might be actually given based on some evidence and observations of the past.

Now we may be facing one of the two constraints that the client could want to impose on us in the contract:

- The client wants the software as fast as we can do it (and preferably even faster)
- The client wants as much as we can do by the date X (which is our business deadline)

Scenario 1

In the first scenario we can determine the predicted time to finish using the formula:

time = (<points> / <velocity>) * <iteration length>

So in our example it would be:

time = (300 pts / 20 pts) * 2 weeks = 30 weeks (or 15 iterations)

If the calculated time is not acceptable then the only factor we can change is the team's velocity. To do that we need to extend the team, however this is not working in a linear way. So doubling the team size does not double its velocity.

Scenario 2

We know that the deadline comes in the next 20 weeks, which are only 10 of our iterations, so we can do only 200 points from the scope. Now it is up to the client to decide what can be removed from the scope without losing the ability to deliver something with the required business value.

Here also (to some extent) we can change the size of the required team to be able to expect higher velocity.

Knowing required team size and target finish time we can finally calculate the main part of the price for the contract – the working time cost:

money = <hrs per iterations> * <no. of iterations> * <people> * <hour price>

In our example we would have:

money = 80 hours * 15 iterations * 4 people * $100= $480,000

So now we have fixed price contract values:

- **price** - $480,000
- **time** - 15 iterations (30 weeks)
- **scope** - 300 story points

Those simplistic calculations are of course just a part of the cost that will eventually get into the contract, but they are also the part that usually is the hardest to define. What we were trying to show here is that the way we work in agile projects can be transferred also to the contract negotiations environment.

There is one more question to deal with. The client asks “why is it so expensive?” This is where negotiations actually start and one thing we want to mention here is that the only factor a software company is able to change is its man-hour cost rate. It is that $100 that we are negotiating. Not the length of our iteration, not even the number of iterations. Our team has no superhero powers and will not start working twice as fast just because we negotiate it this way.

It is the money that we are willing to lose. If we say we can be cheaper it is because we will earn less not because we will work faster.

Tracking Progress and Budget

Now that we have our contract signed (that was easy right?), it is time to actually build the software within the agreed limits. You might have already heard that agile will not solve your problems but it will make them visible. That is why we want to know early how we are doing so that we take proper actions. There is a chance that the magic number (our velocity) from the fixed price contract was a mistake.

Burndown charts are a very common way of tracking progress in many agile projects.

Figure 1 – Planned scope burndown vs. real progress

They are good to visualize planned progress versus the reality. For example the burndown chart from Figure 1 looks quite good:

We are a little above the planned trend but it does not mean that we made a huge mistake when determining our velocity during the contract negotiations. Probably many teams would like their own chart to look like this. But the problem is that this chart shows only two out of three contract factors – scope and time. So what about money?

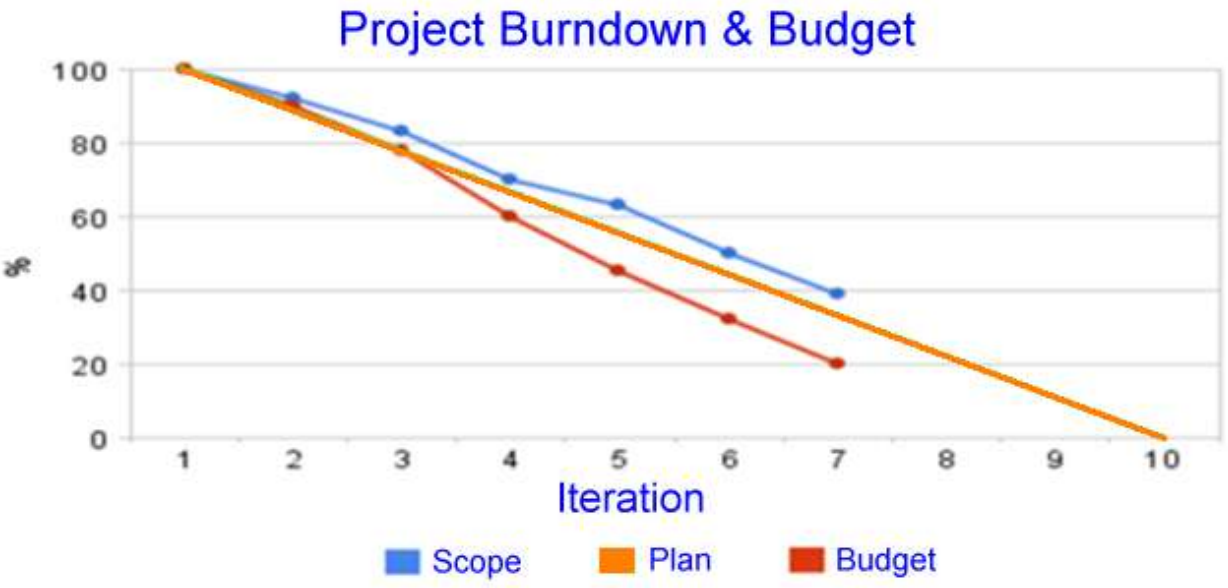

Figure 2 – Scope burndown vs. budget burndown.

The chart on Figure 2 shows two burndowns – scope and budget. Those two trends are expressed herein percent for the purpose. There is no other way to compare those two quantities calculated (one in story points and the other in man-hours or money spent).

The second chart does not look promising. We are spending more money to stay on track than we expected. This is probably because of using some extra resources to actually achieve the expected team's velocity. So having all three factors on one chart makes problems visible and iteration 4 in this example is where we start to talk with the client before it is too late.

To track the scope and budget this way we need to:

- Track the story points completed (done) in each iteration
- Track the real time spent (in man-hours) in each iteration
- Recalculate the total points in your project into 100% of our scope and draw a burndown based on percentage of the total scope
- Recalculate the budget fixed in the contract (or its part) into a total available man-hours – this is our 100% of a budget – and draw a budget burndown based on percentage of the total budget used so far

In our previous examples we had:

- The scope of 300 story points
- The budget of 30 weeks * 4 people * 40 hours a week = 4,800 man-hours

Those would be our totals and spending 120 hours to burn 6 story points would mean that we have burned 2% of our scope using 2.5% of our budget.

Embracing Change

We need to get back one more time to the point where we changed fixed scope into fixed budget. The 300 story points from the previous examples allow us to exchange the contents of the initial user story list. This is one of the most important aspects that we want to achieve with a fixed price contract done the agile way.

Agile embraces change, and what we want to do is to streamline change management within the fixed price contract. This has always been the hard part and it still is, but with a little help through changing the focus from the requirements analysis into some boundary limits early in the process, we want to welcome change at any stage of the project.

The difficulty here is to convince the client that stories can be exchanged because they should be comparable in the terms of effort required to complete them. So if at any point client has a new great idea that we can express as some new set of stories (worth for example 20 points) then it is again up to the client if we are going to remove stories worth 20 points from the end of our initial backlog to make a place for the new ones.

Or maybe the client wants to add another iteration (remember the velocity of 20 points per iteration?). It is quite easy to calculate the price of introducing those new stories, as:

money = 1 iteration * 80 man-hours * 4 people * $100 = $24,000

What still is the most difficult in this kind of contracts is when we find out during the project that some stories will take longer than expected because they were estimated as epics and now we know more about them than we did at the beginning. But still this might not always be the case, because at the same time some stories will actually take less. So again tracking during the contract execution will provide valuable information. Talking about the problems helps negotiating, as we can talk about actions that need to be taken to prevent them instead of talking about the rescue plans after the huge and irreversible disaster.

Earning Trust

All those techniques require one thing so that they can actually be used with the fixed price contract - **trust**. But as we know, we cannot earn it just by describing what a great job our team is going to do for our clients.

Agile principles bring one good idea for solving this problem, so with each iteration you want to build value for the client. More importantly, we focus on delivering the most valuable features first. So, the best way to build the trust of a client might be to divide the contract.

Start small, with some pilot of 2 or 3 iterations (which will also be fixed price, but shorter). The software delivered must bring an expected value to the client. In fact it must contain some working parts of the key functionalities. The working software proves that you can build the rest. It also gives you the opportunity to verify the first assumptions about the velocity and eventually renegotiate the next part.

The time spent on the pilot should also be relatively small when compared to the scope left to be done. This way if our clients are not satisfied with the results, they can go away before it is too late and they no longer need to continue the contract and eventually fail the project.

Agile Meets Fixed Price Contract

Summary

Fixed price contracts are often considered very harmful and many agile adopters say that we should simply avoid them. But most of the time they cannot be avoided, so we need to find ways to make them work for the goal we have, which is building valuable quality software.

Actually, some aspects of fixed factors are even better for agile teams since they are typically used to working within time boxes and that is exactly what the fixed date in the contract (and also fixed price) are -- just time boxes and boundaries. The only thing that actually bothers them is the scope and this article hopefully tried to gather together those ideas on how to deal with that limit.

The intention of this article was not to suggest that agile is some ultimate remedy for solving the problem of fixed price contracts but to show that there are ways to work in this context, the agile way.

About the Author

Michael McCormick founder of MPCS, Inc and Management Professional with 35 years of experience managing over $4 billion in projects for both the Commercial and Federal Government sectors and is a well-known project management (PM) author, consultant, and authority on the subjects of Construction Management (CM), Facility Management (FM), Business Process Management (BPM), Project Management Office (PMO) and Project Portfolio Management (PPM), Risk Management (RM), software development and technology integration.

MPCS Website: www.mccormickpcs.com