

DecisionSpace Infrastructure: Agile Development in a Large, Distributed Team

Marjorie Farmer
Landmark Graphics
mfarmer@lgc.com

Abstract

DecisionSpace Infrastructure was an effort to develop new software in a company where the corporate culture was geared to support old products. The team was large and distributed, and used an agile approach to succeed despite that. In the process, the team helped to lead the company to re-discover how to develop new software products.

1. Situation

Landmark Graphics is a leading developer of commercial software for the oil industry. The company offers a large suite of software products to aid in the discovery and extraction of oil and natural gas, and customers consist of multinational oil companies, national oil companies, and independents.

Originally, Landmark developed a small number of innovative products that revolutionized exploration and established the company. In the many years since, Landmark had grown significantly, and expanded the product line, primarily by acquisition. These acquired products were then modified to make use of Landmark's common data access infrastructure, and Landmark successfully marketed integration as a core business concept.

In the late 1990's, Landmark was a company of approximately 1600 employees worldwide, with an R&D organization of approximately 500, of which half were active developers. There were seven different development offices, in four different countries. R&D was organized into product groups, with a small team assigned to each product and little or no movement between teams.

In the late 1990's, Landmark's core revenue-producing products were all well over a decade old, and were no longer bringing in significantly increasing revenue each year. Landmark executives understood the concept of the Technology Adoption Life Cycle^[1], and recognized that the core Landmark products were reaching their peak earnings and that revenue from these products would

begin to decline within the next few years. They decided that it was critical to the survival of Landmark as a company that Landmark should have new innovative products to offer when revenue from the old products began to decline.

Landmark had few if any new products under development at this time. There had been several attempts within R&D to start developing new products, but these efforts ran afoul of the corporate organization and culture, and failed, generally with a great deal of disruption and pain. DecisionSpace, and the DecisionSpace Infrastructure, was another attempt to create a foundation suite of new products to carry the company into the future.

2. No One Knew How to Succeed

At the start of the DecisionSpace effort, Landmark had just delivered another attempt at large scale new product development, which turned out to not be as commercial as originally hoped. This effort, called Reno, had been more successful than previous efforts, though still not a source of new revenue for the company. Executive management then started up the DecisionSpace project. For this project, they retained most of the same team from the last effort, and gave them the mandate 'This time, do the same thing, only do it right'.

2.1 What We Did

Integration is a core selling point of Landmark software, and any major new Landmark strategy required that integration be a core component. What Landmark needed was a platform on top of which we could build a suite of products, and the previous effort had been a monolithic application, and was not suitable for that purpose.

The first thing we did was to break the large DecisionSpace team into several sub-teams, one of which was the DecisionSpace Infrastructure. This team was to be responsible for delivering the core components that would be needed across multiple applications, and so enable integration. These core components included data access, data visualization, an application framework, and

several middle tier tools such as data analysis and probability analysis. I took on Program Management responsibility for the Infrastructure team.

At the beginning of the project, we didn't have a good idea of basic scope for this new product suite. We had a very high level vision, but no one knew how to translate that vision into the level of detail necessary to develop software. In order to achieve that clarity, we gathered all the DecisionSpace team leaders together, including developers, marketing people, project managers, and domain specialists (this was several dozen people) in a multi-day offsite meeting and designed use cases. This effort included people from both the Infrastructure and the application teams.

In addition, we also had several developers who were tentatively assigned to the new Infrastructure team. Most of them had been involved in Reno, the previous, non-commercial project, and had a lot of strong opinions as to how things could be done better. We called all those people together and established some ground rules.

Many of the Infrastructure developers, including the Lead Architect, were passionately interested in following an agile approach. In particular, they wanted to follow XP (Extreme Programming)^[2], slightly modified.

Then we all on the Infrastructure team decided to do some design and write some code. I started iteration planning immediately and we started work.

The most interesting thing to notice about the above scenario was that all the marketing people, who had the job of deciding what scope the developers should implement, were involved in the use case effort. The Infrastructure effort consisted almost entirely of developers and there was no one involved at all whose job it was to determine scope.

2.2 What happened

The giant scoping effort didn't work out well. It was too unbounded. Also, our marketing people were used to working with mature products, and so expected a product to be of that level of maturity before it could be marketable. The major problems were two-fold. First, the scope that was defined was so large that it would have taken dozens of man-years to implement. Second, there were many important questions around direction that were left unsettled. Because there were no constraints on the problem, there was little information to base decisions on.

The Infrastructure effort was more successful but still had significant issues. The developers started coding, but the client applications had not yet determined their scope, and so the Infrastructure team had no requirements to work from. The Infrastructure team speculated as to what the applications would need, and worked from that. Occasionally, the Infrastructure team would request clarity from the application teams as to what functionality

they needed, but because the applications were still defining their own scope, the answer was always to implement everything. In most cases, the Infrastructure team attempted to oblige.

Since the application teams were still defining scope, and the Infrastructure team was writing code, most of the developers were assigned to the Infrastructure team. Over time, they produced a voluminous amount of code. Many of Landmark's most senior developers were involved in the Infrastructure team. These were people with years of experience in the industry, a great familiarity with our customers and our customers' needs, and significant domain expertise (many of our developers had advanced degrees in relevant sciences). Because of this, most of this code was well designed, and much of it turned out to be useful.

After a few months of this, the Infrastructure began to take shape. There was some basic functionality that an app team could use, and the design had evolved to the point where application developers could understand the intent, and what was likely to be coming. At this point, a small group of the Infrastructure developers, plus a few others, broke out and began working on prototyping applications. They worked closely with the Infrastructure team, in the same code base, and produced some beginning application functionality.

From this, marketing was able to get enough of an understanding of what was possible to be able to define scope, and the application efforts started to make significant progress. Within a few months of that point, several of the Infrastructure developers were moved to the app teams, and were able to bring their expertise with the Infrastructure with them. Also at this time, the application teams began giving requirements to the Infrastructure team, to drive scope decisions there.

As it turned out, while much of the code developed during that initial period was eventually used, there were still large sections that were not used at all and were eventually cut to reduce the maintenance load. The amount of effort wasted was significant.

2.3 Reflections

The big lesson from those initial months was that doing something, even if it's wrong, is better than doing nothing. As it turned out, some of what we did was wasted effort, but most of it wasn't, and we provided the basis that the application teams needed.

Also, the fact we were blessed with our company's top people on the Infrastructure team was a major factor. These people were capable of making good decisions when left without guidance. That involvement of top people was critical.

3. Boundaries and Bite-sized Chunks

In the Reno project, management had given us a hard date to meet, and the team made some strategically unwise decisions in order to achieve the tactical success of making that date. In DecisionSpace, management had learned its lesson, and asked the team to provide the date when they expected they would ship.

3.1 What we did

The various teams set about attempting to define scope so they could come up with a ship date, and found that this scope definition was a nearly infinite task. The Infrastructure team considered that deciding the date was the responsibility of the app teams and didn't worry about it. We would adopt whatever schedule they needed, but since they were the ones delivering the commercial product, we felt it was their job to drive the schedule.

3.2 What happened

The app teams struggled with scope, and the only real development went on in Infrastructure. There was little to no senior management involvement in Infrastructure, because it was too technical for most of the managers to understand. The various teams worked busily, with no schedule determined, for about six months before management had enough.

At this point, management did a reorganization of the larger DecisionSpace effort, replacing the Lead Program Manager and Lead Product Manager (the marketing person). They also assigned a sponsor to the Infrastructure.

As it turned out, we were as fortunate in our sponsor as we'd been in our team members. This sponsor had the technical expertise to understand what we were doing in Infrastructure, and a lot of strong opinions about what and how our team should be delivering. He was a strong advocate of agile development.

The first thing he did was say 'ship'. We did an internal delivery to the DecisionSpace applications of everything we had built so far, which was substantial. We then continued to do deliveries every three months to our client applications. At that point, we had delivered a base that the other applications could build on, and that helped clarify some of their own decisions. Our sponsor also found us a person to act as Product Manager from within the R&D organization, so we had someone besides developers determining scope.

We quickly settled into a new routine around these internal deliveries, which we called Cycles, similar to XP-style releases^[3]. A Cycle was usually about three months long, with four 3-week iterations. We did a round of loose planning as we were finishing up the previous Cycle, used the first three iterations of the Cycle to

develop new functionality, and then the final iteration for stabilization. As the applications became more commercial, we added a longer stabilization period at the end of every other Cycle, which went on synchronously with development of the next cycle.

This provided a lot of clarity for those people outside of the Infrastructure team, as they could see what was actually being delivered. It also allowed the Infrastructure team to consider scope in bite-sized chunks. This turned out not to be such a large issue for us, but turned out to be significant for the application teams.

A few months after Infrastructure started shipping in 3-month Cycles, things came to a head with one of the application teams. They had actually settled on scope and were working on planning, but since there were so many unknowns, it was very difficult to firm up the plan. Eventually, one particular item of functionality, which required work from Infrastructure, became sufficiently well defined that we were able to say that we could not fulfill the requirement in the time frame required. Before that point, all the requirements had been so loosely defined that none of the development team could say firmly that they weren't feasible.

When we said that we couldn't deliver the required scope in the required timeframe, this precipitated a crisis. There was a major meeting of mid-level managers, and they identified the strategic problem of the unbounded and poorly defined scope. The end result was that the application team changed its process to also ship in 3-month Cycles. This forced the marketing people to focus down on a very limited timeframe, and made decision making much easier. Eventually, this approach was rolled out to all the app teams, and they were finally able to extract themselves from the endless cycle of planning.

3.3 Reflections

The scoping effort was trying to bite off something far too large at first, and got lost in it. Once we had limits in place, any limits, we were able to make decisions. The Infrastructure team had been working in iterations since the beginning, so was able to make some progress, but the waterfall, plan-it-all first approach was deadly when there were so many unknowns. We couldn't possibly plan everything, because we didn't know enough, and we couldn't learn what we needed to know without doing at least some execution.

Having a good sponsor really mattered. This is a truism, but it's worth re-iterating. Without that, I doubt we would have succeeded. Also, it's worth noting that since Landmark is a commercial software firm, managers all the way up the line understand the business of software development. When we came up with a crisis, mid-level managers were able to identify the core issues and respond usefully. This would have been much harder in

an IT environment, where the company's core expertise was in some other field.

It would have been nice if our code had been clean enough in the first place so that we didn't need the extensive stabilization, but it didn't work out that way. While we wrote unit tests around new code that we developed, we also made extensive use of older, existing code. This made for hundreds of thousands of lines of code that we had no unit tests for, and we were making use of that code in ways that the original developers had never imagined. Also, the quality of the unit tests we did write wasn't as high as we would have liked.

4. Logistics of a Large, Distributed Team

The DecisionSpace Infrastructure team, at its peak, had forty-one team members, in five different cities, in three different time zones. The team had always felt that agile was the way to go, but the nature of our team presented special challenges. We were fortunate that management had by now been through several attempts to start up new development. The approach management was using this time around was 'throw top people at the problem and hope they come up with a solution'. They let us find our own way, rather than forcing process on us from above.

4.1 What we did

Landmark and the Infrastructure team had no experience implementing an agile methodology in a large, distributed team, and very little experience even with smaller teams. However, we very much did understand that we weren't going to succeed by using a traditional approach. The problem we had to solve was too big, and the requirements were too fuzzy. Given that, we started adapting the various agile methodologies as best we could. Our approach paralleled and took ideas from the approach that Cusumano and Selby recommend in *Microsoft Secrets*^[4]. The big areas that we knew we wanted to cover were:

- Iterations
- Unit testing
- Continuous builds

Also, we recognized that we were going to have special problems with communication, and needed to make an extra effort there.

As a start, we broke the larger team down into multiple sub-teams. The sub-teams were generally co-located, and were oriented around a particular set of functionality. That at least bought us the benefit of small co-located teams for much of our effort, and allowed us to focus on the intra-team issues that affected the team as a whole.

For the rest, we made use of the agile approach of 'continuous learning'. That is, trial and error.

4.2 What happened

Trial and error worked fairly well, although we certainly managed plenty of the 'error'.

Iterations turned out to work quite well with a large team, with very little adjustment necessary. We had a couple of technical people that were able to understand the larger picture. These people acted as the customer for each sub-team's iteration review. The end result was that each sub-team's efforts meshed in with the project's overall goals.

Unit testing was rocky at first, as not all the developers on the team considered unit tests to be a valuable use of their time. The thing that turned out to make the difference there was when we started publishing metrics every week. These metrics showed unit test coverage by package, and graphically illustrated which areas of the software were building up unit tests and which weren't. Since the team leadership was pushing unit tests aggressively, and since the graphics made it clear which developers were writing unit tests and which weren't, unit test development picked up nicely after that.

Build stability had been a significant issue during the previous release (Reno), enough so that everyone on the team understood the need for a change. We addressed this issue by implementing continuous builds that fired off whenever anyone checked in code. If the build did not complete successfully, the build script then sent an e-mail to the developer that checked in the offending code, plus numerous other people, every ten minutes until the build was fixed. Peer pressure proved effective, and build stability improved significantly.

Effective communication was an ongoing effort, and broke down into two basic approaches. The first thing we did was to formally organize communication opportunities, to replace the informal communication that would have occurred naturally in a smaller, co-located team. Basically, this meant scheduled conference calls. We had a weekly leadership team meeting, a weekly testers' meeting, a weekly architects' meeting, plus iteration reviews with each sub-team every three weeks. In addition, the team members very quickly grew used to conference calls, and became comfortable requesting additional conference calls whenever an issue needed to be discussed.

It turned out that while conference calls were not as effective as face-to-face communication, and didn't provide as natural an opportunity for information exchange, they were sufficient in most cases. The one area that conference calls proved not to be sufficient was for architectural design. In that case, we ended up having regular multi-day meetings of our architects, all in the same room. In the early days, we had those meetings as often as every two or three weeks. Even now, we still have them once every quarter or so.

The other approach we took to communication was information distribution. The idea there was that by publishing a large quantity of the information that team members needed, that would reduce need for communication, and therefore reduce the pain that team members suffered due to the large, distributed nature of the team. The most effective distribution mechanism was the company intranet. Several team members put up web pages, but the most useful turned out to be the one run by the Configuration Management person, which contained pointers to the builds, various metrics, instructions of various sorts, and a large collection of other useful information.

We also attempted to institute various processes and standards, to make the large team work smoothly. That approach worked less well. The team members fought the introduction of processes that they couldn't see value to, and in most cases were fairly successful in fighting these efforts off. The most spectacular failed process was the attempt to introduce coding standards, which generated such argument that it was never implemented.

4.3 Reflections

We found a way to make a large, distributed team work with an agile approach, but there was no question we would have been more efficient if we'd been co-located. However effective conference calls could be, they still weren't as effective as face-to-face conversation, and information meetings in the hallway.

It made a difference that the initiative to take an agile approach came from within the team. The level of commitment was significant. Interestingly, I believe that it made the team less tolerant of errors and problems rather than more. Whenever something wasn't working as smoothly as some team member thought it ought to be, we, that is, the team leadership, would hear in great detail and at great volume exactly what the problem was and what we ought to be doing to fix it.

I think that vocal openness was a significant contributor to our success. Given how much we were learning as we went, we really needed that input to help us improve.

Peer pressure worked well. Since the team as a whole had committed to an agile approach, pressure to cooperate was significant, as was pressure on any team member who wasn't living up to group expectations.

5.0 Overall Reflections

When we started, we all brought out our little white Kent Beck books and started arguing over how much Extreme Programming we could implement. In reality, rather than following Extreme Programming, we were

much more closely aligned with the fundamentals of the Agile Manifesto.

According to the Agile Manifesto^[5], the four principles of Agile Development are:

- Individuals and Interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Of the four, I think we did best on the second and the fourth. Given the large, distributed nature of our team, I think we did exceptionally well on the first, though we did have to impose some process. On the third item, I can certainly say we didn't do any contract negotiations with our customers, but that had more to do with the lack of customers. We did work very closely with our client applications, and that was very much in the spirit of an agile approach.

We had several major factors that contributed to our success:

- Our team consisted of some of the top people in the company
- We were permitted to find our own way
- We got support from management when we needed it, especially in the area of our sponsor

I think all three of the above factors were necessary for success, and that we would have failed otherwise.

The end result of the DecisionSpace effort is that Landmark is now producing new software again, not only in DecisionSpace, but in other areas as well. Revenue from new products is rising, and we have a significant quantity of new products in our pipeline. Only time will tell, but right now it looks like Landmark's effort to re-learn how to create new software has succeeded.

References

- [1] Moore, G.A., Crossing the Chasm, HarperBusiness, 1991
- [2] Beck, K., Extreme Programming Explained, Addison-Wesley, 1999
- [3] Beck, K. and Fowler, M., Planning Extreme Programming, Addison-Wesley, 2001
- [4] Cusumano, M.A. and Selby, R.W., Microsoft Secrets, Touchstone, 1998
- [5] Agile Manifesto, www.agilemanifesto.org