

# Agile Development in the old economy

Géry Derbier  
Solystic  
gery.derbier@solystic.com

## Abstract

*As part of the delivery an automated hub for a postal operator, the Solystic company has to build a complex and feature rich Information System that supports a highly automated process with multiple intricate sub-processes and exceptions. The effort has several challenges:*

- *It is business critical for the customer, and the output of the project will give the customer a leading position;*
- *It is the first time Solystic is managing a complete system project, although its mother company Northrop Grumman had previous experience of this business;*
- *The program is one of the first business of the Solystic company in the international field;*
- *It is a fixed price, fixed time contract with a short time frame.*

*To face all these challenges, the software development group in charge of the Information System has adopted a number of agile practices and techniques to manage the project. The major project settings are adapted from Alistair Cockburn's Crystal set of methodologies, SCRUM and Jim Coplien's work on organizational patterns. This paper presents the findings and lessons learned by the team and its manager.*

## 1 Introduction : project overview

The project objective is the delivery by Solystic of an automated hub for the french postal operator La Poste Courier International (LPCI), that is the integration and ramp-up in a large facility of :

- A complex assembling of automation equipment provided by several companies and sub-contractors ;
- A feature rich Information System supporting a complex process with multiple intricate sub-process and exceptions.

Some key figures illustrates the dimension of the project :

- the facility will employ 700 people working on 38000 m2 over two floors;

- 200 international flight departures are handled every day;
- more than 150 truck arrivals and departures every day;
- about 30 different process flows are supported (a flow is an end-to-end processing of some kind of mail product);
- about 80 different workplaces equipped with one of the 20 different applications developed;
- deployment of the solution within 5 months;
- more than 250 tables in the database, about 2000 C++ classes and 150000 executable code lines.

This paper is about how of the software development group in charge of the Information System, which is a major part of the overall project, managed the software development effort.

The project had several challenges :

- It is business critical for the customer : deregulated market will dramatically exacerbate competition in a very near future ;
- It is the first time the customer, La Poste Courier International, relies on a integrator to provide a turn-key system ;
- It is the first time Solystic is leading a complete system project, although its mother company Northrop Grumman had previous experience of this business ;
- Though Solystic has a long experience in the automation of domestic mail processing, the Roissy program is one of the first business of the Company in the international field ;
- It is a fixed-price, fixed-time contract with a (perceived) short time frame.

The development effort amounts to an average of 20 persons IS team over 18 months for the construction of the Information System ; the overall project team has an average 60 persons.

As the manager of the Software Development Group in charge of the IS, I have extensively studied the field of Software Engineering. My approach to management of software development is mainly influenced by Alistair Cockburn and Jim Highsmith's writings on the Agile methodologies, Gerry Weinberg's classics on software quality management, Ken Schwaber's writings on Scrum,

Jim Coplien's work on organizational patterns, the practice of Aikido and improvisational music.

Right from the beginning, we thought this project had the characteristics an agile approach could match:

- Overall vision and mission clearly articulated;
- Complexity of the business, high level of tacit-knowledge expected (the as-is level of automation is very low, so a lot of things ingrained in the practices and the head of people), pretty high number of interfaces to external information systems (a significant part of which with poor level of written documentation);
- Lot of things to learn, particularly how to grow a good working relationship with the customer;
- Business critical for the customer.

## 2 Agile techniques and practices used

The Agile techniques and practices the group used and adapted came mainly from [1, 6, 10]. This part presents the major agile techniques and practices we have used. A more detailed description of some points is made in the following sections.

The communication settings:

- A lot of face to face communication : we managed to get the team collocated on the same floor in a building with 3-4 persons per offices, grouped by subject matter. With very rare exceptions and/or interruptions, the developers are dedicated to the project.
- A Scrum like meeting (structured around the three basic questions asked to every member of the team during the meeting) is held in front of a whiteboard used to track the current iteration progress (more details below).
- Iteration retrospective: a half-day meeting at the beginning of each iteration.
- Customer involvement: during iteration planning, requirement elaboration, end of iteration review.

The planning techniques:

- One-month structured timeboxed iterations ;
- Milestone and iteration planning : items planned in an iteration are 5-10 days use cases or pieces of use case; iteration planning done on the first day of the iteration is a half-day meeting involving the whole team ; the milestone planning is done with simple tools as described in [1, 7].
- Incremental specification with use cases, *à la* Alistair [5], to manage the level of precision. The planning is mainly based on use case.

The development practices:

- Unit testing: use of CppUnit;
- Frequent integration: build at once a week on a dedicated build environment with efforts to reach the daily build and smoke test;
- Continuous testing: internal delivery of identified software release every week tested by 2-3 people during the following week;
- Some level of automated regression testing with a COTS product;

The team structure:

- Light definition of roles (akin to Crystal Yellow) ;
- The software development team comprises an average of 20 people structured in two layers :
  - Solution layer : three project managers are responsible for delivery of the solution to the customer. They collaboratively elaborate the specifications of the system and do project planning. Each project manager was more specifically leading the development of a functional area. The test team belongs to that layer (2 full time testers)<sup>o</sup>; the PM are doing testing activities during the second part of the project (about two third of their time). One PM was the overall project leader ;
  - Component layer: the developers of the components of the system. This layer has to consider reuse of components (product line development). This layer has four functional sub-teams(7, 3, 2 and 2 people) and one technical sub-team (2 DBA in the database team).

## 3 Findings

### 3.1 Finding #1: how we adapted some agile practices

Holding a « scrum meeting » was still possible with the complete 20 persons team. Some accommodations were necessary. We hold it every other day for approximately 30mn. The functional teams members (that is application developer) are asked the 3 basic questions, the database team talk when they think it necessary. The team seems to be comfortable with that way of doing. The scrum meeting is done in a dedicated room, in front of the iteration plan made of Post-It notes sticked on a whiteboard. Both practices work very well together.

The Post-It notes contains the title of the use cases or items to be tackled and the associated estimate in days. Each Post-It note is sticked in front of the name of the person it is assigned to. Based on the information provided by each member, the conductor of the scrum meeting (similar to the ScrumMaster) moves the Post-It

notes in successive columns to reflect the status of the task : not started, started, integrated and ready to be tested by the test team. It is an example of Gerry Weinberg's Project Poster [9] and Alistair Cockburn's information radiator [1].



Figure 1. Scrum meeting in front of the whiteboard

I had to explain the 'barely sufficient' concept used there. I consider the whiteboard mainly as a reminder, a support for discussion during the initial planning meeting

and the scrum meetings : the most important thing is what is said in the meeting. However, there has been some discussions in the group about the content of the Post-It notes : some found there were not representing all the work done, some others felt more comfortable. I think among the first group are people who are more sensitive to public recognition of their contribution to the project.

The collection of Post-It notes is basically what we used to track the progress during an iteration and is the finest level of planning and tracking. We did not track the iteration with a iteration profile and a burndown chart [6]. We considered the whiteboard efficiently conveyed the status information to the team (a and to chickens).

However we had a form of scrum product backlog (several of them in fact) in order to track for the estimate at completion (the company tracks projects progress through Earned-Value analysis) and the CompletionHeadroom [10].

During the planning session we focus on the realization of features (use cases or piece of use cases). The teams collectively estimate what can fit in the iteration, there is no detailed estimation for the database developments or support activities. The team is comfortable with that. We used the Yesterday's Weather principle [4] to plan each iteration.

On another whiteboard, the pending problems are listed, thus giving everyone a chance to remind the ScrumMaster to deal with some of them. We found useful to write on the whiteboard the decisions that must be taken outside the scrum meeting, then in the next meetings ask for what decisions has been taken : this ensures everyone is in synch.

On some other part of the whiteboard is the list of all the things the group has decided to try or improve during the last retrospective meeting.

### 3.2 Finding #2: how we are going to adapt some agile practices.

At the time of this writing, the project is entering the system integration phase. This phase is typically very turbulent. We will have a mix of development and integration activities. We are considering turning to one-week iterations to be responsive to the system integration vagaries and constraints.

## 4 Lessons learned

### 4.1 Lesson #0: it works, but...

My management agrees that the approach used seems to provide more control over the course of the project. Most of the customer team is satisfied with the high level of interaction we have provided so far. We have delivered intermediate milestones. However, at the time of writing this paper, the complete system integration is under way and there is still a significant effort in front of us.

But so far, we have clearly seen that the adoption of the approach is essentially a question of changing attitudes, and that it is a difficult task. It took several months for most of the team members to feel at ease with the approach.

For me, as a manager, the Agile approach was first a way to make the team learn quickly and to always be kept awoken. This is the difference between being on guard (agile way) and in guard (process oriented). However, not a lot of people are comfortable with uncertainty and agility. It is still true one year into the project. As an example, all people are not comfortable with a (too) light definition of role ; it is perhaps truer in a (rather) large team.

I even more strongly believe the cooperative game [1] is a valuable model to understand software construction. More people around me are now convinced that people is by far the number one factor that influences a software project.

### 4.2 Lesson #1: it takes time before people fully understand all the power of the scrum meeting.

I consider the scrum meeting to be one of the main building block of the method we used. During the retrospective meeting, the discussions about suggestions about how to improve the process reached more than once to the conclusion : « speak during the scrum meeting, it is there for just that purpose ». But everyone is not at ease with talking publicly about what is going wrong in its job, and/or to say he welcomes help to solve the problem at hand. It necessitates courage, and it is necessary to give a lot of energy to develop the proper culture to inspire confidence in the group (note : I am the manager of the group and also the « ScrumMaster »). Courage is not something you can detect during hiring.

I regretted every time I had a bad feeling during a scrum meeting and did nothing about it. To handle uncomfortable situations, I would systematically hold a meeting with the simple following format :

- First, address the emotional part by giving each participant the opportunity to express how he *feels* about the situation ;
- Second, try to state the problem and find a solution.

The format of the scrum meeting alone is not sufficient. I proposed the scrum meeting technique to another development group in the company. But the group did not hold the meeting in front of the board. The practice did not caught up.

### 4.3 Lesson #2: embarking the agile way really needs solid people skills.

At the very least, you should enjoy to take the risk to interact with people [2]. Agile development is a cooperative game requiring lot of human interactions. As I favored direct face-to-face communication, I had to learn how to debug the communication process and invest quite a lot of time in that activity, coaching and teaching people how to communicate in a tolerant, assertive way.

Beside knowing people work better in some ways than others [1], it helped me to have in mind that people communication and behavior have some common bias among which :

- the fear of being ridiculous;
- do not show one's errors;
- do not cause pain to others;
- perfectionism;
- the fear of being manipulated;
- everyone should love you.

### 4.4 Lesson #3: short iteration can make you short-sighted.

It is usually hard to have long-term view and most developers can only think in the short term. Short iteration reinforces the risk to be shortsighted. Counter-balancing practices must be put in place. I think architecture review is a natural one because it addresses problems within a large scope (overall project and business). Blitz planning is another one, though we managed to do this only once so far.

I think that we did not fully implement the planning meeting at the start of the iteration. It is obviously another good occasion to address the long-term view. The thing we could do:

- Have a better practice of assessing the current status at the beginning of the planning session;
- Have the senior management involved in the meeting.

#### **4.5 Lesson #4: install automated regression testing practice as early as possible.**

I think we started a bit late with the automation of the functional tests. The unit testing started almost at the beginning of the project and it works pretty fine now, although it is not a piece of cake (it is far more than just take CppUnit and use it).

I consider automated unit testing is a place where the manager can put pressure to get it. I weekly ask the team to give the percentage of code that has automated test and the number is made public.

#### **4.6 Lesson #5: set up and discuss configuration management right from the beginning.**

As configuration management is a collaboration practice, it is one core process and deserves very high attention. In our experience, it has been the first hot topic discussed in retrospective meeting.

#### **4.7 Lesson #6: holding a retrospective meeting is one key building block. It is hard. Holding it every iteration is even harder.**

It is one of the main subject of interest for me now : learn how to better hold that kind of meeting. I think it is at the heart of getting the Agile attitude. We have tried the simple format described in [1]. Our experience shows that the discussions went partly structured, partly unstructured. In fact, we found it was necessary to use that time to discuss very general and fundamental subjects about software development (what is it, how can we estimate accurately, ...) so that the team can have a common understanding of what is going on. The discussions inevitably went about values. Discussing about fear (of being wrong, ridiculous, ...) was to me one of the most important subject that has emerged (cf. lesson 2).

#### **4.8 Lesson #7: the timeboxing principle is fundamental.**

Hold it firm. Explain more than twice that the timeboxing is marginally about time but really about focussing and forcing tradeoff decision.

Parkinson's law is a hard fact. It is difficult to find the appropriate level of pressure in order to meet project deadlines without setting up a death march project. On the other hand, software developers are notoriously optimistic and tend to overcommit. So the end of the iteration is the

hard part. The most important thing is to repeatedly make the people understand that the end is a duration and not a event, and to understand what happens during the end. It is why right from the beginning I have adopted the timeboxed iteration as a 3 phase cycle [8] : definition (1-2 days) – construction – consolidation (a few days).

The timeboxing principle is the number one thing the agile approach has brought that my management believes in.

#### **4.9 Lesson #8: personal recognition needs counters self-organizing team.**

As the manager of the group, I saw people needing assurance that their contribution is valuable and recognized (mainly by me). Trying to have a team self-organize can lead to the problem that people don't see their personal contribution is recognized (by the manager). I handled it through attention during the scrum meetings and personal conversation with each developer.

#### **4.10 Lesson #9: the estimation process needs to be truly shared.**

The estimation process and techniques are not well known among developers. If you want a self-organizing team, the team must truly share the estimation process. Estimation is done more frequently and the uncertainty is higher in an agile approach. It is the root of risk analysis and commitment taking.

I should have explained and discussed more and earlier the estimation process.

#### **4.11 Lesson #10: some things that did not work or caught up.**

I set up a wiki with the following aim: write short descriptions of the practices on the upper half of pages with comments, links, etc... in the lower part where anyone could add things. The wiki did never really get alive. I have been almost the only one to write. Are the people satisfied with what was exchanged during the scrums, the retrospective meeting and the hallway chats ? I guess so.

Not so much external people came to play the «chickens» during the scrum meeting. However, those who came regularly found it useful.

We did not completely succeed with the daily build (the daily build is not currently followed by smoke tests). However, trying to achieve it served as an indicator to improve the architecture, the configuration management and the capability to release the complete software in a few days which seems sufficient. That is why we relaxed on the practice.

#### **4.12 Lesson #11: what I found myself doing as a manager.**

I mainly found myself being a coach that serves as a feedback mechanism to the development team. In the classical model, information goes from the team upward to the manager who makes (most of) decisions. To me, the Agile approach is somewhat different: the manager makes strategic decisions and acts as an external consultant questioning socratically the team for information and giving feedback in order to help the team understand what they are doing and make the good tactical decisions. More than ever, the manager should not do in place of the team: this can bring difficulties in some cultures.

As every manager, I had to take care of the exchange of information between the team members. But it is even more important as we rely on face to face conversations. For example, some people are not naturally verbose and others are shy of asking for more explanations.

I found one of the major roles I have was to keep the team motivated, have ownership and move forward (e.g. by reminding the whole picture).

I found myself handling (a lot of) psychological conversations. I found the number one question I had to ask the team members to be: « what do you fear ? ». I have found the most demanding situations are:

- During the scrum meeting, to help people manage the peer pressure when talking publicly about their progress (and especially to speak out the problems);
- During the retrospective meeting.

The majority of things I found I had to do need (very) solid people skills. I have learned a lot, and I found myself teaching several members of the team the techniques found in [2].

I think the agile community can usefully elaborate on the manager's role.

#### **4.13 Lesson #12: have practices to manage architecture.**

It is important to have a good practice of architecture review. I did not find in books or articles a particularly 'agile way' to do it. Jim Coplien reports the Borland Quattro Pro team (one of the most highly productive teams observed) discussing architecture in daily meetings. However, Scrum [6] recommends keeping the daily meeting short by focussing on synchronizing the team members.

We have found useful the scenario-based software architecture evaluation techniques described in [3] and in the CMM reports on architecture evaluation. The approach reminds me of Kent Beck's statement (in a mail on the XP list?) that he can turn any requirement (whatever functional or non-functional) into a user story. We used this approach only to hold some review. I think the project would have benefited from a more systematic use of the technique. I am looking for other agilists' experience reports on this subject.

#### **4.14 Lesson #13: the development team should learn time management principles and techniques.**

Once you become a manager, you are usually sent to time management courses, to improve your efficiency. Every one should benefit from that. I found myself teaching the team the principles and techniques I use to manage my time, to handle interruptions and so on. I think an agile team can benefit from learning that early.

### **5 Conclusion. Lesson #14: tentative agile 101**

If I had to choose the three first things to setup, discuss and teach this would be:

Things to setup:

1. The timeboxed iteration structured with the core meetings: planning, scrum meeting, user review and retrospective;
2. The (almost) daily build;
3. Automated unit and functional test.

Things to discuss:

1. What is it to do software development?
2. The estimation and planning process;
3. The build and delivery process.

Things to teach :

1. People skills;
2. Time management;
3. Technical skills.

## 5.1 Author's musing

Some of my favorites quotes (personal translations) :

“Errare humanum est, perseverare diabolicum ”

“There are no technique that does not work, there are only techniques that do not work the way you would like them to work. ” - an aikido teacher.

“A army leader demands victory to the situation, not to his subordinates. To command people, the true method is to use the avaricious and the dummy, the wise and the brave and to give each one responsibilities that fit to him. Don't ask anything to the untalented” – SunTzu (500 BC).

“What is simple is necessarily wrong, what is not is useless” – Paul Valéry.

“Perfection is achieved not when there is nothing to add but when there is nothing to remove” – Antoine de Saint-Exupéry.

## 5.2 References

- [1] A.Cockburn, *Agile Software Development*, Addison-Wesley, 2001.
- [2] R.Bolton, *People skills*, Touchstone, 1979.
- [3] P.Clements, R.Kazman, M.Klein, *Evaluating Software Architecture*, Addison-Wesley, 2002.
- [4] K.Beck, M.Fowler, *Planning Extreme Programming*, Addison-Wesley, 2001.
- [5] A.Cockburn, *Writing Effective Use Cases*, Addison-Wesley, 2000.
- [6] K.Schwaber, M.Beedle, *Agile Development With Scrum*, Prentice Hall, 2002.
- [7] E.S.Andersen, *Goal Directed Project Management*, Kogan Pages, 1984.
- [8] J.Stapleton, *Dynamic System Development Method*, Addison-Wesley, 1997.
- [9] G.Weinberg, *Quality Software Management*, Dorset House, 1997.
- [10] J.Coplien, *Organizational patterns*, .