

# From Anarchy to Sustainable Development: Scrum in Less Than Ideal Conditions

Isabelle Therrien  
Pyxis Technologies  
itherrien@pyxis-tech.com

Erik LeBel  
Pyxis Technologies  
elebel@pyxis-tech.com

## Abstract

*In early 2008, Pyxis Technologies was contracted in the third of a series of projects distributed between Canada and France. This project dwarfed its predecessors in scope and complexity. In over more than a year, it involved more than 30 people in 3 cities. With contributors separated by distance and time zones, organizations and expertise, the challenges faced ranged from organizing and coordinating the teams with Scrum and managing the evolution of the product. Through attention to problems and a willingness to adapt, the team has steered clear of any project-ending problems and managed to maintain a sustainable development pace. This is our story and what we've learned along the way.*

## 1. Introduction

### 1.1 The project

This is the story of an ambitious, complex, and distributed project with contributors from 5 different companies working full time for over a year at the time of this writing. The contributors are spread across 2 continents with a time difference of 6 hours.

Our story focuses on 3 recurrent themes:

- Organizing Scrum across multiple sites with near 20 people
- Managing code complexity and maintaining architectural cohesion
- Working efficiently with 2 satellite teams: operations and interface design

This is a large-scale project with a simple but ambitious target: inventing tomorrow's TV. With this service, you can create your own TV channel, program its content, and share it with your community.

### 1.2 The context: the companies' first collaborations

From July 2007 to February 2008, Pyxis took part in two small-scale projects in collaboration with CVF – online multimedia and later BeTomorrow. Our organizations had been hired for their respective expertise: development of rich clients and applications on mobile devices (BeTomorrow) and development of complex applications using Agile engineering practices (Pyxis). For both projects, CVF was the company contracted by the end-client, and CVF subcontracted to BeTomorrow and Pyxis. All participants were considered full members of the team; the roles were not defined according to their respective employer.

In March 2008, a third project was presented to us in Paris. Based on our previous collaborative successes, we felt we could keep working with a distributed team. Even with a guesstimated 30% overhead associated with working remotely, we encouraged the clients to carry out the project with a distributed team. Our arguments were:

- Communication between the organizations was excellent. A relationship of trust was established.
- Our expertise complemented one another. There was a significant difficulty in hiring experienced people in Paris.
- Though young, the technologies to be used had delivered in the previous projects (Flash for mobiles, Flash for PCs, Grails and Groovy for the backend).
- There was a sense of urgency to start the project and launch the product as quickly as possible.

## 2. Project start-up

### 2.1. Sprint Zero, making hard choices

A team of dedicated Product Owners (POs) welcomed Isabelle in Paris on March 6, 2008. She was responsible for driving the project's Sprint Zero as the

ScrumMaster. The POs were from the client, an entity of the Orange Group, whose official mandate was to create innovative services rapidly and efficiently. Since the idea was still a vague one at the time, they were hoping that leveraging the rapid ROI promised by Agility would allow them to meet the challenge.

It is in this atmosphere of confidence and optimism that the 2-week kick-off took place. We were not afraid of anything. Two developers from BeTomorrow (Bordeaux), the ScrumMaster from Pyxis (Montréal), and the team of POs from the client (Paris) met in Paris and drafted the project charter, a non-exhaustive, prioritized, and estimated backlog of over 200 epics and user stories, a release plan and operating rules for the entire team.

At Pyxis, we conduct Sprint Zeros as regular sprints. We assemble a backlog of “project start-up activities”, we prioritize it, plan it, and at the end of the sprint we review deliverables. Each morning of this Sprint Zero, we stand up for 15 minutes and talk about the objectives and activities for the day, and ask for a measure of the team members’ satisfaction level.

The main goal of these two weeks was to reduce the scope of the project to a realistic subset of functionalities. The approach that best helped the POs to prioritize their backlog was to associate a purchasing cost in Euros to each story point. It was easy for them to remove functionalities when faced with their implementation cost. Their budget was not unlimited.

After establishing the minimal project scope, we agreed that 3 sprints of 3 weeks were required to produce an application that could be tested by the users. An important commitment was asked from the team: 9 weeks of work with 6 developers to deliver 72 story points and build the core service.

## **2.2. First three sprints, gaining trust: April-May 2008**

Erik joined on day 1 of the first sprint. Along with another developer in Montréal, their first challenge was finding a working phone to conference in with our POs in Paris and developers in Bordeaux. They lost 1 hour in the first planning, because they didn’t set themselves up for a distributed conference.

Falling back on previous experiences, we rapidly organized the activities of the distributed team:

- Sprint planning of 4 to 6 hours with webcams
- Daily scrum of 15 minutes by phone
- Sprint review and retrospective of 4 hours with webcams

Our meetings were held in the afternoon for the French and in the morning for the Canadians.

The communications with our first external partner, the *interaction design* team, were generally good. They

contributed to Sprint Zero by helping us define the project vision. Then, they adapted to the incremental development by rapidly providing the first screen drafts while developing the comprehensive long-term vision of the project.

Quickly we tried out and identified the tools that best helped us collaborate across the distances:

- Skype took the lead as the communication tool of choice. It allowed us to setup permanent chat rooms for different collaborative groups. Obviously, this required equipping the teams with enough headsets to be able to start multiple distributed conversations as needed.
- JIRA<sup>[1]</sup> and the GreenHopper plugin<sup>[2]</sup>, for managing our backlog. This tool helps us manage a backlog of over 1000 cards with views that allow different teams to manage their sprint content and burndown charts individually or collectively.
- A wiki to store all forms of documentation and reference material. This area proves useful for orientation documentation for new team members.
- Group mailing lists.
- Team and project calendars (Google).
- A shared spreadsheet (Google docs) appropriately named the “White board”.
- 2 telephone bridges and hands-free telephones for conference calls.
- Later in the project, we developed an in-house custom videoconference application (WarTV) to allow video-conferencing on three or more sites, available at all times on our servers.

The idea is to quickly settle on an accepted set of tools and make them available permanently so that the team can concentrate on its main concern: developing the client’s application.

On the technical side, these first sprints saw us investigating different candidate technologies. We chose to use some bleeding-edge technologies, hoping to leverage the wealth of features they offered and promised to soon support. It seemed like a good idea at the time. This choice eventually caused us some grief. The technology did not mature and stabilize as quickly as we hoped, and the size of the application would eventually prove difficult to manage within the framework.

Since the integration environment was not a priority at that time and no one was assigned to set one up, we installed a temporary monoserver integration system. This temporary server proved indispensable for the whole first year of the project, until April 2009.

In the middle of the second sprint, Isabelle went back to Montréal (Canada) to work with her colleagues as a developer. The role of ScrumMaster was transferred to one of CVF's personnel (finally someone is available!) in Paris.

In the first three sprints, we delivered 63 story points. We considered it a success: with our transparency and skills, we had gained the client's trust. It was the beginning of a long relationship that would no longer be in the shadow of a contract based on delivered story points.

We were beginning to feel the effect of 'delivering points at any cost'. By the end of the third sprint, we were 9 developers. We had already accumulated a technical debt, and the number of defects gave us food for thought: we decided that we would never again cut corners to deliver as many points as possible.

Never again? Hum!

### 3. Scaling up

#### 3.1. Growing the team: June-September 2008

We received the go-ahead to scale the team to 15 members. Development continues as we gradually integrate new members. We are finding our meetings to be unproductive and noticing certain "smells":

- Sprint planning meetings, daily scrums, and sprint reviews are becoming interminable.
- The team is fragmenting into smaller groups.
- Team members are not feeling committed, responsibilities are diluted.

We decide to split the team in two. One team in Bordeaux and the other composed of members in Montréal and Bordeaux, and eventually Paris. Each team has its own sprint backlog and ScrumMaster, but both teams share a common backlog and POs. The sprint planning meetings are done separately by groups, but a single sprint review is done for both teams. Isabelle returns to her role as ScrumMaster for the distributed team. The other ScrumMaster is assumed by one of the original technical members of Sprint Zero. He is located in Bordeaux with his team. To coordinate the teams, we introduced a daily "Scrum of Scrum" in which the ScrumMasters and the POs hold a daily meeting after the individual team daily Scrum. The daily "Scrum of Scrum" is conducted with the standard three questions.

During summer, the teams grow on all sides. Members are added in Bordeaux, Paris, and Montréal. The new members in Paris do not feel part of the team and lack adequate contact with other developers to feel integrated. We make some adjustments 2 sprints later by moving the CVF members to Bordeaux. All French team members are now together. We decide to continue the hiring of developers in Bordeaux.

Concerning the design, the results of the focus groups held at the end of June 2008 made the clients change their mind regarding the artistic direction. We expected that the impact would be minor. After all, we

are an Agile team. We embrace change, don't we? The reality proved quite different: 43% of the functionalities' development costs are allocated to the development of purely graphical components (29% of the project's total cost). Changes to the design had a major impact on the project costs... and on the morale of the developers.

For 2 sprints, our team of skilled graphic developers was on standby waiting for the new design. They decided to take advantage of the break to get familiar with the back-end and learn Groovy. This brought up a new challenge. We had long cultivated and nurtured the code to the best of our abilities and worked hard to maintain a high degree of test coverage and code clarity, flexibility, and decoupling. Our colleagues from the GUI side had different ideas on code structure, and did not have our background in testing. We needed to coach them to maintain code stability in this dynamic language landscape. This proved harder than expected. Some conference calls and presentations were held, as well as peer reviews, but the difficulty in efficiently communicating ideas across vast distances made this very time-consuming and proved inadequate. The importance of automated testing was imparted to the GUI team, and upon their return to the Flex world they started experimenting with automated test frameworks.

A new role emerged: one of the developers in Bordeaux manages communications with the design team. Previously, an entire development team would get tied up in meetings with the design team, attempting to communicate back and forth their needs and ideas. Now, only one person will be in charge of these communications. This helps to maintain organization in the otherwise chaotic exchange of emails, and helps developers make the most of their day.

There are various exchanges with the operations team and important discussions on the application architecture but due to organizational restrictions, no one from this group has been dedicated to our project and needs.

Over the summer, the team size has doubled, but our velocity remains nearly constant. Our search for ways to improve our productivity leads us to plan the 'semaines bordelaises'.

#### 3.2. The 'semaines bordelaises': September 2008

Finally, we all met in Bordeaux! The impact of these weeks together was beneficial in many aspects, the most important one being the development of a relation based on trust. Having heard of the 'Five

dysfunctions of a team<sup>[5]</sup>, we knew that without a solid trust relationship, the team could never perform well.

This trip has been an incredible experience. In Bordeaux, we paired with our colleagues from CVF and BeTomorrow. They were welcoming, friendly, and most of all engaging. During these weeks, we made good resolutions. We decided to make promising changes regarding our ways of doing things, and particularly we developed a keen sense of being a real team.

The need to have more flexible teams was felt for several reasons. In our team, we did not want to put one specific person in charge of a module. We did not want to be unprepared if someone had to leave the team. We also wished that everyone would have the opportunity to work on everything. It was a challenge especially with the size of the application and variety of technologies involved. Furthermore, we wanted to sustain the team spirit among all 15 developers. We also wanted all team members to spend some time with a co-located team. We knew the team building theory and also knew that we would never have a team with the "Performing" status<sup>[6]</sup>. However, the benefits we expected seemed to compensate for this suboptimal situation.

Unrestrained, the artistic vision leads to technical complexities and significant costs. To help our POs make realistic choices, we proposed to involve developers earlier in the process for the design maintenance. Later on, we pushed this principle further with anticipation workshops that allowed them to get developer feedback on technical effort during the design phase.

A new tool improved the communications between the two continents: the WarTV was born. It was also around this time that we learned that one of the POs would be moving on.

## 4. Continuous improvement

### 4.1. Keeping the rhythm: October-December 2008

Regarding the daily "Scrums of Scrum", we rapidly felt a sense of disquiet: they are only being used to report progress to the PO. A few smells appeared:

- ScrumMasters are viewed as information hubs in spite of themselves. The daily Scrums are not used for the teams' coordination; they are used to keep the PO informed of the team's progress.
- It is difficult to feel a real overall commitment: the ScrumMasters feel they are responsible for the commitment of their respective team

To rectify the situation, after discussing with colleagues, we decided to have developers participate

in the daily "Scrums of Scrum". To optimize the meeting, we try to limit the number of messages to be transmitted from one team to another and not to report on everything said during the meeting. The progress is not very important since we can track it on the burndown chart. However, the PO finds it difficult. She likes to have her daily report...

At this point, we had our first recurrent interactions with the operations team. Though again the assigned resources changed frequently, they were in charge of building integration, pre-production, and production platforms. We had weekly calls with them to discuss our evolving needs and discuss our late deployment problems. The complexity of our infrastructure needs combined with their security restrictions made deployment very difficult. Deploying the application became a continuous preoccupation, one that occupied at least on developer full time through to the beginning of 2009.

### 4.2. Smelling bad: January-March 2009

Winter 2009 proved hard on the project. Several situations that seemed sustainable during the previous periods rapidly escalated and became painful. This period was marked by several departures (one person moved, another resigned, and Erik accepted another mandate as a coach), replacements, and a code audit.

As the application started going through its first real testing in a production-like environment, bugs started to emerge. Though we always knew there would be some issues, the application was not as stable as we hoped. Problems were so rampant that during a whole sprint one team delivered a whopping zero point; they spent so much time trying to clean up the accumulated problems of the neglected code maintenance. "Fix later" had finally caught up with us.

Some developers started the full time job of fixing bugs and refactoring the code to hammer out some of the structural kinks in the code base. How could we avoid accumulating so much dead weight? With the first deployment in production taking place so late in the project, several realities were discovered at that time. This lack of visibility was rapidly adjusted for with a rapid estimation of the work to be done out of this very simple formula:

If <b>ND</b> = the proportion of the work 'Not Done' yet to complete the deployment of a stable application in production (e.g. SQL migration, performance tests, etc.)
If <b>SP</b> = number of Story Points accomplished
If <b>TBD</b> = Estimate of what has yet To Be Done
<b>ND x SP = TBD</b>

In our case, we evaluated ND at 33%. We came to this conclusion by evaluating the weight of each item required by our complete (ideal) definition of 'Done',

and divided this weight of what is not included yet in done by the total weight of the ideal Done.

If we had included these incomplete tasks in the backlog as we went along, these tasks would have been easy to evaluate, and the POs would have had a better view of the work to be done. It is conceivable that this knowledge would have influenced their priorities as we neared the date of the beta launch.

### 4.3. Towards an integration team pattern

Adding developers to the team did not increase the velocity as much as the PO expected. First, the PO was getting buried with work. Predictably, she became a bottleneck to team productivity. The team members got used to compensating for her absence by prioritizing the backlog themselves; explaining stories to one another and re-scoping user stories in mid-sprints. Accordingly, we lacked a global vision for the future of the project.

Second, because of the constantly changing teams, it was difficult to know who had the expertise needed to help prepare backlog items for later on. The developers were not consulted early enough for their technical feedback, and sadly several tasks essential for the success of the project were never prioritized: developer tools, automated end-to-end testing, code refactoring postponed in favor of delivering functionalities. The team also felt the need to devote some time to revising the architecture of the application but could not easily justify this to the PO during the prioritizing of the backlog.

Finally, as mentioned earlier, we tried not to have module specialists. This remained true, but we started to feel the need for technical leads for the different modules. They would be helpful in resolving conflicts arising in the architecture.

We decided to restructure to adopt “Integration Scrum” as proposed by Ken Schwaber in “The Enterprise and Scrum”<sup>[3]</sup>. Here are the changes we brought to the organization of our team:

1. We added an integration team to the project. This team is responsible for ensuring the coherence and usability of the collective contributions of all teams. It is the main point of contact for the PO. This team is in charge of the following tasks:
  - Assure architectural coherence and compliance with project conventions
  - Build supporting tools for the developers
  - Fix bugs that prevent the application from being usable in production
  - Deploy the application in the production environment
2. We added two new POs, one for each development team. These POs have a greater

availability than the primary PO. They are responsible for specific themes in the application's functionality.

The daily “Scrums of Scrum” were impacted. We felt the need to keep them for coordination purposes, but we took care of discouraging the reporting tone (the POs assigned to the teams took care of transmitting the team's progress) and enabling an efficient communication between the teams, and with the POs. The daily “Scrums of Scrum” then focused on different questions (MCQ):

- Do you have **M**essages for the other teams?
- Do you respect your **C**ommitment?
- Do you have **Q**uestions for the other teams?

### 4.4. Working towards the first beta testers: April-June 2009

The current situation is pretty encouraging. With the integration team dedicated to helping the other teams focus on functionalities and the POs dedicated to their teams, a lot of impediments were removed.

We still have a hard time communicating with the infrastructure team though, and still pay the technical debt related to the late start of the installation of the production environment. Now, we manage this debt by making it visible to everyone and at the end of each sprint, measuring the amount of paid (or accumulated) debt. We also created a “technical backlog” prioritized by the team for these tasks.

We talked about the urgency for starting this project at the beginning of this paper. Well, as you can see, at the time of this writing, we have not released the service publicly yet. Looking back, now, we wonder if we should have waited a little bit to start, or whether it was a good idea to start right away

So, what have we learned... so far?

### 4.5. Lessons learned, by themes

*About size and distribution:*

- Organize face-to-face meetings quarterly. As Ken Pugh states in his presentations, “whether you do it or not, you will pay for it”<sup>[4]</sup>.
- Favour small teams of 4 to 6 over larger ones. The distribution of the team means that communication is already one of the biggest hurdles. Each additional member of the team increases the channels of communication that need to be maintained.
- Use videoconferencing for discussions over email and other forms of communication.
- Do not underestimate the impact of integrating new members. Not only do new members rarely

contribute to team velocity during their first weeks, but they can slow the team down.

*About code complexity and architectural cohesion:*

- Cultivate a vertically modularized architecture corresponding to the themes and split the teams according to those modules.
- Value clean code: it is the best assurance against degradation of team velocity over time.

*About external groups:*

- When possible, include a member of each external group within your team.
- We have often read that specialized roles are created as needed. It is important to watch for such needs, but only create them as needed.
- Do not take for granted the cost of deploying and testing your application. If you do not have a production-like environment on which to deploy and test your application, then there is a portion of the post-development activities defined in your 'Done' that are not being completed. Make that uncompleted work visible to your POs otherwise they will be in for an unpleasant surprise.
- Do not start developing as long as you do not have the means to release the application.

#### 4.6. The advantages of distributed teams

Are there some benefits to developing with a distributed team?

First consider that with a project spread over several cities it is possible to recruit from a larger pool of resources; it can be easier to find skilled people as needed. This is often one of the reasons that distributed teams come into being.

Now consider the time difference. While it can be problematic if the difference is too great, a time difference with a comfortable overlap can open the door to some very productive days. The overlap is necessary to ensure knowledge transfer, team confidence, hand-off of work from one team to the next, and communicating issues or problems to the other team. With a 6-hour time difference, we find that both groups have a half-day of overlap in which meetings are held, and the groups collaborate on tasks, and both groups also have a quiet period in which they can focus on their individual tasks. This half-day of quiet time is ideal even for projects that do not deal with a time difference. It also causes a half-day of dead time between sprints in which developers can catch up on some activities that they have been putting off, be they refactoring, reading, or updating documents.

## 5. Conclusion

Keep your eyes open for common problems: trust problems, mentoring needs, people hiding inefficiencies, lack of visibility for the PO, technical debt that starts to grow unnoticed in the code.

Talk about your project, publish articles, blog about it and write an experience report to share your experience. Give and receive: especially if you think everything goes well, do not be afraid to be measured or audited. Evaluate every piece of advice that is being generously given to you by colleagues or evaluators.

Be wary of silver bullet solutions and "secret sauces". Look to them for inspiration, but not salvation. Often you will be working with different ingredients, and will need to find ways to make the best use of them.

Do not be afraid to make mistakes (everyone does), but be ready to implement changes when you see that something is not working.

***In short, be attentive to how things are going and be creative in looking for solutions.***

## 6. Acknowledgements

The authors want to acknowledge everyone at Pyxis for their inspiration, help, and support throughout the project; Nathalie Gilbert for her help in the translation of the early drafts from French as well as the final review. Thanks to our friends and colleagues at BeTomorrow and CVF – online multimedia for their passion, and contributing to a welcoming team spirit. Thanks to our client for entrusting us with this project, and thanks to our shepherd Rachel Davies for her encouragement and excellent suggestions.

## 7. References

- [1] JIRA, Atlassian, <http://www.atlassian.com/software/jira/>
- [2] GreenHopper, <http://www.greenpeppersoftware.com/>
- [3] Ken Schwaber, *The Enterprise and Scrum*, Microsoft, 2007
- [4] Ken Pugh. *Managing Distributed and Global Teams*. Software Best Practices Conference, Boston, 2007. (Quote from slide 59).
- [5] Patrick M. Lencioni, *The Five Dysfunctions of a Team: A Leadership Fable*, Jossey-Bass, 2002
- [6] Bruce W. Tuckman, *Developmental sequence in small groups*, Psychological Bulletin 63 (6): 384-99