Final thesis

Practical implementation of SCRUM and associated practices

by

Etienne Girot

LIU-IDA/LITH-EX-A--13/054--SE

2013-10-14
Final Thesis

Practical implementation of SCRUM and associated practices

by

Etienne Girot

LIU-IDA/LITH-EX-A--13/054--SE

2013-10-14

Supervisor: Andreas Fröberg
Examiner: Erik Berglund
PRACTICAL IMPLEMENTATION OF SCRUM & ASSOCIATED PRACTICES

Etienne Girot
Spring 2013

Paris, October 6, 2013
Abstract

Eventhough Scrum is nowadays widely known in the Software Industry field and its theoretical frame extensively described in the litterature, its implementation is far from being straightforward. As a matter of fact, the litterature describes a *meta-process* that new practionners must adapt to their project specific constraints. However, this practical aspect is crucial and very scarcely tackled. To that extend, this thesis work describes the difficulties we faced while putting Scrum into practice and how, through the study of the project contextual factors, the insight of a Proof Of Concept and the support of a couple of agile practices, we worked it out.
Thanks

to Erik Berglund for the quality of his advice, the helpful questions he raised and the great freedom he accorded me in his guidance.

to Yann Gensous for having been an available, supporting and truly pedagogue mentor.

to Laurent Balagué and Nicolas Meary for having been comprehensive and involved superiors.

to Hassan Allybocus and Yinjia Yang for being a particularly sympathetic and friendly colleagues.

to the whole Formetris staff for its great sympathy and its agreeable welcome.
Contents

I  Introduction 8

II  Background 9

1  Agile & Scrum 9
   1.1  Agile .......................................................... 9
   1.2  Scrum ....................................................... 10
       1.2.1  Overview ........................................... 10
       1.2.2  Theory .............................................. 10
       1.2.3  Roles ............................................... 11
       1.2.4  Ceremonial ....................................... 11
       1.2.5  Artifacts .......................................... 13

III  Project description 14

2  Formetris company: Roots & evolution 14

3  A new module for new needs 16
   3.1  From customization back to generalization* ................. 16

IV  Early steps 17

4  Things just got real 18
   4.1  Initial hesitation: documentation overload ..................... 18
   4.2  Take a deep breath and jump into Scrum with both feet ......... 19
       4.2.1  Define Scrum ceremonial ................................ 19
       4.2.2  Define monitoring tools ................................. 22
       4.2.3  Technical choices and gross conception* .................. 22
   4.3  Initial backlog ............................................ 24

5  Early steps sprints’ retrospection 27

V  Incorporation of scrum-compatible practices 28

6  Incorporation of velocity 29

7  Incorporation of planning poker 31

8  Sprint reviews & retrospections 32

VI  Recommendations and Conclusion 33

9  Qualitative observations 34

10 What has worked out 34

11 For the next sprints 35

Etienne Girot  Spring 2013 4/61
Annexes

1 The 360-degree feedback process 37
2 Sprint 1 - Planning 39
3 Sprint 2 - Planning 40
4 Sprint 3 - Planning 41
5 Sprint 4 - Planning 42
6 Sprint 5 - Planning 43
7 Sprint 6 - Planning 44
8 Sprint 7 - Planning 45
9 Sprint 8 - Planning 46
10 Sprint 9 - Planning 47
11 Sprint 10 - Planning 48
12 Sprint 11 - Planning 49
13 Sprint 12 - Planning 50
14 Sprint 13 - Planning 51
15 Sprint 14 - Planning 52
16 Sprint 1 & 2 - Retrospection 53
17 Sprint 3 & 4 - Retrospection 54
18 Sprint 5 - Retrospection 55
19 Sprint 6 - Retrospection 56
20 Sprint 7 & 8 - Retrospection 57
21 Sprint 9 & 10 - Retrospection 58
22 Sprint 11 - Retrospection 59
23 Sprint 12 & 13 - Retrospection 60
24 Sprint 14 - Retrospection 61
# List of Figures

1. Easier with a model .................................................. 8
2. Iterative and incremental approach ................................. 10
3. A Sprint ........................................................................ 12
4. The product backlog .................................................... 13
5. The sprint board ........................................................... 14
6. Burndown chart ............................................................ 14
7. Kirkpatrick’s four level of training evaluation ...................... 15
8. Formetris evaluation process ......................................... 15
9. Custom actions weaving ................................................ 17
10. Project contextual factors .............................................. 20
11. Initial sprint design ..................................................... 21
12. Initial release plan ....................................................... 21
13. Initial backlog population ............................................ 22
14. Importance of starting well ............................................ 27
15. Velocity charts and trends for sprints 1, 2 and 4 ............... 30
16. Velocity charts and trends for sprints 1, 2, 4, 5 and 6 ........ 31
17. A scrum team using planning poker ............................... 32
18. Velocity charts and trends for all sprints ......................... 33
List of Abbreviations

- **SaaS** - Software as a Service
  is a software delivery model in which software and associated data are centrally hosted on the cloud. SaaS is typically accessed by users using a thin client via a web browser. (wikipedia)

- **TDD** - Test Driven Development
  is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards. (wikipedia)

- **PoC** - A Proof of Concept
  is a realization of a certain method or idea to demonstrate its feasibility, or a demonstration in principle, whose purpose is to verify that some concept or theory has the potential of being used. (wikipedia)

- **PO** - The Product Owner
  represents the stakeholders and is the voice of the customer. He or she is accountable for ensuring that the team delivers value to the business. The Product Owner writes (or has the team write) customer-centric items (typically user stories), ranks and prioritizes them, and adds them to the product backlog. (wikipedia)
Part I

Introduction

Scrum is nowadays so popular that most Computer Science students as I am, study it at University. Even-though I was familiar with the topic, the first move I undertook when I have been brought to implement it was to read a book that precisely details it over hundreds of pages with the hope to assert my knowledge. When finished, I knew rather extensively what Scrum was all about and what practices it prescribes. Neverthe-less, I did not find what I was looking for, that is, confidence in how to proceed in order to put it in practice. Well, maybe I missed something in that book, maybe it needed time to digest the info or maybe it was just not the best book for that matter... Fair enough, I decided then to read more about Scrum and eventually feel ready. Among these readings, I came to study a couple of documents that intended to be rather practical. I discovered thus with enthusiasm how Scrum can be coupled with dozens of other Agile practices but, at the same time, with perplexism, how indecisive I still was toward how to implement it. The benefit advertised around Scrum is how its empiricist essence fits with the reactive nature or human beings. As a matter of fact, it is considered simpler for human beings to achieve a creation through iterative rework of tangible drafts rather than keeping on the abstraction level all the way through. Scrum creators bet that its practionners will, imperfections after imperfections, improvements after improvements, end up with a satisfying practice of Scrum meeting their needs. I am not in a position to discuss human nature thoroughly but would modestly venture to add that, as reactive as they may be, humans, in my opinion tend to be reflexive as well. I refer here the propensity to learn by example and reluctance to take risks when no others took those risks before. After a little while I understood that the very reason I did not feel confident while reading books about Scrum was their lack of a model to mimic. This paper describes thus the route we followed to get over it and will hopefully constitute one of the testimony I would have appreciate to find before leaping into Scrum.

![Figure 1: Easier with a model](image)
Part II

Background

1 Agile & Scrum

In this section, we will introduce the keys points of Agile software development and Scrum. Most of what will be presented here is well known in the field of software. Hence, the readers who feel familiar with these concepts are invited to jump to section III.

1.1 Agile

Before the last 20 years, software development projects were framed by the so-called "waterfall" model and its variations (such as the V-model). These methods prescribe rather linear processes where the whole software product is achieved through requirement elicitation, design, coding, test and deployment. However, it has been observed that customer’s needs and the expression of his needs evolves significantly as the software development progresses and that traditional waterfall-oriented models were hardly flexible enough to respond to it. As a matter of fact, going all the way through the different development steps (and their extensive documentation phase) until delivery before being able to correct the project orientation is slow and costly. The lack of flexibility in the model is pointed out as one of the most common software development failure factors [8]. To address this flaw of the heavy weighted traditional methods, software engineering paradigm has evolved toward incremental development cycles in the late 50’s and progressively gave birth to the Agile methods.

In 2001, a group of 17 software developers published the Manifesto for Agile Software Development[3] which reads as following:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
> 
> Individuals and interactions over processes and tools
> Working software over comprehensive documentation
> Customer collaboration over contract negotiation
> Responding to change over following a plan

> That is, while there is value in the items on the right, we value the items on the left more.

And is based on the following 12 principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
• Working software is the primary measure of progress.
• Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
• Continuous attention to technical excellence and good design enhances agility.
• Simplicity—the art of maximizing the amount of work not done—is essential.
• The best architectures, requirements, and designs emerge from self-organizing teams.
• At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Most specific methods that follow these principles present a certain number of similarities such as lightweight planning, time-boxed activities and self-organized cross-functional teams. The particularities of Scrum, which figures among these methods, will be described in the next paragraph.

1.2 Scrum
1.2.1 Overview
Scrum is a framework that intends to help overcome uncertainty and complexity while producing highest value product in shortest time-to-market. Scrum being a framework, not a method or a process, prescribes very few practices. It is therefore easy to understand and lightweight. On the other hand, it requires rather important efforts to find out how to fill the gaps in methodology that Scrums does not prescribe. It may thus be truly difficult to adapt it to real projects.

In the most condensed description I managed to formulate, Scrum would consist of a team (with different roles) that participates to a ceremonial (periodically scheduled events) to produce artifacts (tangible piece of work) following rules.

1.2.2 Theory
Scrum [15] is empirical. It is based on the idea that knowledge come from experience, from evidences. Therefore it encourages making decisions with light of what is known and tangible. Accordingly, Scrum uses an iterative and incremental approach that intend to increases risk control and predictability.

![Figure 2: Iterative and incremental approach](image)
1.2.3 Roles

The scrum team consists of a Product Owner (PO), a Development Team and a Scrum Master whose functions and responsibilities are described below.

The Team

The scrum team model is designed to optimize flexibility and productivity. Therefore, the Scrum team imperatively is:

- **self-organizing**
  
The team chooses itself how work is to be divided, scheduled and achieved.

- **cross-functional**
  
The team members together gather all competencies required to accomplish the work independently from others not part of the team.

Scrum teams deliver products iteratively which gives maximal room for feedback and shortest lead time for adjustments. Furthermore, delivered product, should at all times, be partially usable (even if unachieved) and named potentially shippable.

The Product Owner (PO)

The Product Owner is accountable for:

- Defining and prioritizing tasks to achieve the product (i.e. define and maintain the Product backlog)
- Validating the quality of accomplished work
- Ensuring that the team members share a consistent vision of the product under development.

The Scrum master

The Scrum master is in charge of:

- Ensuring understanding of Scrum by the team and the respect of its principles.
- Buffering interactions with those outside the team and removing impediments against its productivity (e.g. budget considerations, resources allocations).
- Coaching the team in self-organization and cross-functionality.
- Finding techniques likely to improving the process.

1.2.4 Ceremonial

The scrum ceremonial[2] prescribes a number of events (meetings) that encourage regularity and minimize logistics-related issues as well as superfluous time spent in meetings. Events are time-boxed: Every event has a maximum duration allocated in the schedule that is to be rigorously observed. In doing so, the workflow is expected to be fluid and meeting easy to schedule.
The sprint

A sprint is a time-box during which the team works toward creating a potentially shippable product. A sprint consists of:

- a sprint planning meeting which defines the sprint goal and task planning (Sprint backlog).
- daily scrum meetings which help synchronizing the work and daily monitor progression.
- development work during which the team is working to achieve the assigned tasks.
- a sprint review meeting that states on task completion and goal achievement.
- a sprint retrospective during which the sprint process is inspected and improvement are proposed.

Prescriptions:

- Sprints last maximum one month.
- No changes in sprint goals are allowed during the sprint.
- Sprints length is fixed and regular from sprints to sprints.
- Sprints may be cancelled by the PO under exceptional conditions.

Sprint planning meeting

Sprints planning meetings are time-boxed events during which sprints goals are decided. Sprint goals specify what is to be delivered at the end of next sprint and how the tasks are articulated. The PO presents to the team the prioritized upcoming tasks, or Product Backlog, from which the development team selects a number of its highest prioritized tasks. The number of selected tasks is based on a team self-made forecast concerning its own capacity of achieving these tasks before sprint end. The set of selected tasks together with a plan to achieve them is called Sprint backlog.

Daily scrum

The daily scrum is a 15-minute time-boxed event during which the team members synchronize their activities and plan their effort for the next 24 hours. Therefore, each of the team members successively answers the following three questions:

- What has been accomplished since the last meeting?
- What will be done before the next meeting?
- What obstacles are in the way?

The daily scrum is intended to improve communication, share a common vision of project progression and structure, encourage quick impediments identification and removal as well as fasten decision-making.
Sprint review

During the sprint review, the scrum team, together with the stakeholders, inspects the increment and adapt the product backlog.

- The PO claims the completed and uncompleted tasks during the past sprint.
- The team members discuss about the problems encountered and how they were solved.
- The team demonstrates the accomplished work.
- The PO presents the current state of the product backlog.
- The attendees collaborate on the upcoming tasks definition and refactor the product backlog.

Sprint retrospective

The retrospective meeting is dedicated to team self-inspection with respect to relationships, process and tools. Its purpose is to identify positive practices and harmful dysfunctions as well as proposing an action plan to implement improvements.

1.2.5 Artifacts

Product backlog

The product backlog constitutes a priorized list of tasks that contains everything necessary to achieve the product. A product backlog evolves throughout the whole product development. The product backlog may be re-prioritized and its tasks re-estimate or re-defined as the specifications gain in clarity. As a matter of fact, as the development goes, the customers know better what he/she wants and the team understand better his/her actual needs. Initially, only best-understood requirements figure in the product backlog.

![Figure 4: The product backlog](image)

PO and team members may collaborate to define, order and clarify tasks. However, the PO has the final say concerning task definition and prioritization while team members make the final estimate.

Sprint backlog

A sprint backlog is a set of selected items from the product backlog which achievement is scheduled during a sprint. The selection is made by the development team based on their own work capacity forecast. The team updates its sprint backlog all along the sprint with task status changes and estimated remaining work.

Tasks are displayed on a sprint board where they are (hopefully) moved from the "To do", "In progress" and "Done" columns successively. Tasks may be deleted if they are considered obsolete and new tasks may be added by the team if needed. Only development team members are allowed to perform changes in the sprint backlog.
Monitoring tools

It is recommended to sum the total remaining work estimates at every sprint review to assess progress and delivery date plausibility. However, it is possible to do it more often. Especially when using a project tracking software, charts may be generated periodically and automatically. The most used chart is the so-called burndown chart:

The burndown chart tracks the sum of remaining work estimates and compares it with the ideal linear effort that would suffice to achieve sprint tasks on time.

Part III
Project description

2 Formetris company: Roots & evolution

In 2004, Laurent Balagué and Nicolas Meary founded Formetris. Having both worked many years in the Industry in contact with training programs conception and monitoring, they could note Industry’s growing need for unbiased ways of evaluating them.

The company they have built fulfills this needs for three different stakeholder categories:

- **Companies:** by helping them to select accurate training programs according to their improvement areas and measuring their impact.
• **Trainees**: by conveying their impressions on trainings to their management board and ensuring that the selected programs truly develop their skills.

• **Training companies**: by giving them valuable feedback on their services and favorizing their alignment with customers’ needs.

To that end, the company has been offering a web-based evaluation platform that provides the trainees with questionnaires, processes their answers and publishes reports. In parallel, the company issues benchmarks that rank training provider companies according to different criteria.

In matter of training evaluation field, the studies of Kirkpatrick highlighted four significant levels:

![Figure 7: Kirkpatrick's four level of training evaluation](image)

1. **Reactions**: "How was the training?".
2. **Learning**: "What did you learn?".
3. **Transfer**: "What are you using in practice?".
4. **Result**: "How does the performance benefit?".

The Formetris evaluation process complies with this model, in that it, traditionally, consists of the following steps:

1. Ten days after the training session date, a first *on-the-spot* questionnaire is sent out to collect trainees’ immediate reactions and cover the first two Kirkpatrick’s levels of evaluation.

2. Two months later, a second questionnaire is sent out. This time, the purpose is to collect actual impacts of the training over time in the trainee work habits. This second questionnaire provides feedback concerning Kirkpatrick’s last two levels.

Additionally a (nd if the customer wishes so), a pre-questionnaire can be sent before the training session to help the trainer estimate the best approach to teach and state on the progression made.

![Figure 8: Formetris evaluation process](image)
3 A new module for new needs

Some of the paragraphs will describe the very heart of the project that supported this work. It may, from time to time, get rather technical. In some extend, I consider this technicality necessary to render the genuineness of our Scrum implementation. The outcome of this work is nevertheless independently understandable and I invite the reader to peck in these paragraphs depending on his or her interest of that matter. Such paragraphs will be labeled by a "*".

3.1 From customization back to generalization*

On the figure presented above (see figure 8, page 15), we can note the linear character of the traditional evaluation process in use at Formetris. As well, its different steps are independent one from another (i.e. the questionnaires cover different areas of the evaluation and their answers do not rely on each other). As Formetris gains ground on the training evaluation market, the customers keep coming up with specific needs. These needs result in slight deviations of the classical evaluation process and range from a simple extra questionnaire up to complex links between several questionnaires (involving interdependency and information transfer from questionnaires to questionnaires).

A common example consists of a pre-questionnaire that allows a respondent to fill in another person contact info (name, email, role) to invite him or her to answer a survey. Technically, the implementation of such pre-questionnaire goes beyond the simple retrieval and analysis of answers. It requires to create user profiles as well as issuing mail invitations on-the-fly. It is a whole different process.

So far, the solution adopted was to "weave" custom actions before/after questionnaire loading as well as before/after answers submission (see figure 9, page 17).
Classical answer submission process:

- Respondent opens questionnaire URL in browser.
- Display questionnaire to respondent.
- Respondent answers questions and click "submit" button.
- Save respondent answer in database.
- Display confirmation message "answer submitted with success".

Answer submission process with weaving connectors ( ) and custom weaved custom actions ( ):

- Execute custom pre-load actions.
- Execute custom pre-save actions.
- Execute custom post-save actions.
- Display questionnaire to respondent.
- Respondent answers questions and click "submit" button.
- Save respondent answer in database.
- Display confirmation message "answer submitted with success".

Example: Create user and invitation out of answers submission

- Execute customer #1 pre-load actions: "Check if respondent SAP Id is valid".
- Execute customer #1 pre-save actions: "Check if to-be-created invited user SAP Id is valid".
- Execute customer #1 post-save actions: "Create user with submitted answers and invite him to next questionnaire".
- Display confirmation message "answer submitted with success".

Figure 9: Custom actions weaving

To introduce flexibility in the process, specific actions for each customer specific need has been developed. The specific action were however almost never reusable for a customer which they are not specifically designed for. Specific actions for customer 1, that for example uses the answer to question 3 as invited user "email" cannot be reused for customer 2, which uses instead question 5 to do so.

Eventhough not interchangeable these custom actions are often asked to perform similar tasks:

- Link answer values from questionnaires to questionnaires (e.g. create pre-filled user profile)
- Create invitations to a questionnaire
- Send out emails (e.g. survey report)
- ...

To favour the resuability of custom actions and reduce the development effort, Formetris has decided to integrate a workflow engine to the evaluation platform which would handle these process specificities smoothly.
Part IV
Early steps

4 Things just got real

The days immediately following the reception of the project specifications were full of hesitation. It is during that very period that I really wished I could rely on a model to give the project a clear, even if not perfect, kick-off. That is why, I am willing to share our experience and some of its outcome with hope that new Scrum practitioners could use them to back their own implementation.

4.1 Initial hesitation: documentation overload

Within that early beginning, I was reading much about Scrum methodology. Thought it gave me a pretty comfortable feeling of being guided, I had the feeling not to know where to start. I found myself running in circles around the same following principles:

1. Define your product backlog.
2. Prioritize it.
3. Estimate the high priority tasks.
4. Define first sprint log according to team capacity.
5. Run the sprint.
7. Meet at sprint end, review, re-prioritize, refresh charts, improve and start over.

Which basically does not provide very much extra information than the famous Scrum process diagram (see figure 3, page 12), if not a verbose description over hundreds of pages.

- Scrum prescribes the use of a prioritized product backlog and the initial estimation of its tasks. What should figure in it? Since we do not really know yet the approach with which the problem is to be tackled (e.g. off-the-self solution or home made solution), it was difficult to elect tasks and to estimate them even more.

- Scrum prescribes the use of short-time boxed sprints. How long?

- Scrum prescribes the planning of sprint according to the team capacity. What is our team capacity?

Before actually starting to use Scrum, I needed to confort myself by answering all of these pending questions and started thus with what appeared the most straightforward: clarifying the tasks. To that end, I ran without second thoughts into the usual project plan. I spent about a week drawing up lists of stakeholders, functional and unfunctional specifications as well as formal UML use cases. Satisfied, I went on. I carried out a rather extensive study of off-the-self workflow engines and wrote down the pros and cons of each. A pleasant feeling of reliable progress first filled me. To monitor my progression, I regularly reported to Yann (our Product Owner for the project and more generally Formetris’ software development director). Each of these regular meetings fed consistently ny To Do list with adjustments and modifications to be made in my documents. Rapidly, it became clear that, keeping my documents up-to-date from one meeting to the next, would need me working at least 70% of my time on refactoring them and only 30% on actually going forward. Furthermore, I noticed I seldom was, if ever, using my documents to present my work to Yann and, in return, Yann would very rarely need to consult or check them. Obviously, persevering in such a way to define our usage of Scrum would lead to an excessive time-to-market. Besides, implementing a methodology by starting to violate one of its key principles “working software over comprehensive documentation” [3]
appeared as a non-sense. Hence, I could not afford waiting hypothetical answers to my interrogations before actually using Scrum. Still I needed these answers! But then, I understood that they did not necessarily have to be perfect, just ”good enough” to start.

4.2 Take a deep breath and jump into Scrum with both feet

4.2.1 Define Scrum ceremonial

At this point of the reflection, I started to look rather for qualitative indicators than for complicated calculations (that would, by miracle, be applicable to our case). I selected sub-set of applicable projects contextual factors from the study of Scott Ambler[1] and I drew up an archetype of ours.

- **Project nature**
  - **Criticality**
    
    *Medium*: The software produced is not safety critical as for example an automotive embedded software would be. However it requires rather high quality standards to maintain Formetris high reputation among its customers, its competitive edge.
  
  - **Change frequency**
    
    *Potentially low*: The needs are well known and clearly expressed by the Formetris team although the means to realize them are really uncertain at this early stage of the project.
  
  - **Deployment type**
    
    *Great degree of control*: The platform mostly work according to the Software As A Service paradigm. Hence we are free to deploy updates often (as opposed to stand-alone software clients) and work on short releases, short sprints.

- **Software development status**
  
  - **Architecture stability**
    
    *Highly unstable*: The project being brand new and its technical aspect yet unknown, the software architecture is likely to be modified very often. This consideration might force us to adopt medium-sized sprints to balance our effort between architecture adjustments and new feature implementations.
  
  - **System age**
    
    *Brand new*: The lack of medium/long-range visibility on tasks encourages to work with short increments as well as it prevents us from defining elementary tasks.

- **Team**
  
  - **Size**
    
    *Very small* (up to 4 members): The small size of the team facilitates setting up meetings and shortens their durations. It encourages us to take advantage of this flexibility and use short sprints.
  
  - **Engineering skills**
    
    *Unbalanced*: Though I possess certain skills in object oriented programming and a certain knowledge of web-based application architectures, I had no experience with PHP, Zend and Javascript, technologies which the whole project relies on. However, the other team members have had great mastery on them. Therefore my early needs in matter of education was expected to cause a reduced work capacity at first.

  - **Geographical dispersion**
Any: All the team members possess a desk in the same room except Yann that sits in the room next door. Meetings would then require almost any logistics and allow for a high interaction frequency.

- **Organization**
  
  - **Business model**
    
    *Adapted and reliable*: Formetris has been working closely with its customers for years, its business model is very compatible with short iterations and the development team should not face impediments of that kind.
  
  - **Management style**
    
    *Adapted and reliable*: Yann encourages the team to express organizational considerations and takes them into account when possible. He works with a participative rather than directive management style which is very compatible with scrum principle of team self-organization.

Once the project so characterized, I intended to qualitatively identify impediments to a ”perfect” Scrum process. A ”perfect” Scrum process consists, among others, in as-short-as-possible sprints. Still it is prescribed to respect weekly periods to encourage the association between meetings and days (e.g. ”Friday means is retrospection day”). In short: ideally one-week sprints[2]. The perfect Scrum prescribes as well breaking tasks down to elementary sub-tasks (e.g ”implement a given function”, ”fix a given bug”) rather than working with composite tasks (e.g ”implement a given feature”, ”refactor code”).

<table>
<thead>
<tr>
<th>Project nature</th>
<th>Sprint length</th>
<th>Task length</th>
<th>Team capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Criticality</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td></td>
<td></td>
<td>might require medium</td>
</tr>
<tr>
<td><strong>Change frequency</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Potentially low</td>
<td></td>
<td>allows long</td>
<td>allows long</td>
</tr>
<tr>
<td><strong>Deployment type</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Great degree of control</td>
<td>allows long</td>
<td></td>
<td>encourages short</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software development status</th>
<th>Sprint length</th>
<th>Task length</th>
<th>Team capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture stability</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Highly unstable</td>
<td>might require medium</td>
<td>prescribes short</td>
<td>reduces</td>
</tr>
<tr>
<td><strong>System age</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brand new</td>
<td>encourages short</td>
<td>makes fine breakdown and estimation harder</td>
<td>reduces</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Team</th>
<th>Sprint length</th>
<th>Task length</th>
<th>Team capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very small (up to 4 members)</td>
<td>encourages short</td>
<td>encourages short</td>
<td></td>
</tr>
<tr>
<td><strong>Engineering skills</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unbalanced</td>
<td>prescribes short</td>
<td>prescribes short</td>
<td>reduces</td>
</tr>
<tr>
<td><strong>Geographical dispersion</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>allows short</td>
<td>allows short</td>
<td>increases</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Organization</th>
<th>Sprint length</th>
<th>Task length</th>
<th>Team capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Business Model</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adapted and reliable</td>
<td></td>
<td></td>
<td>increases</td>
</tr>
<tr>
<td><strong>Management style</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adapted and reliable</td>
<td>allows short</td>
<td>allows short</td>
<td>increases</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Do not constrain nor influence choices</th>
<th>Sprint length</th>
<th>Task length</th>
<th>Team capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>influences positively</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>might influence negatively</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>influences negatively</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 10: Project contextual factors
Accordingly with these considerations, we decided to start on a one-week sprint base. Plus, we were warned on possible difficulties regarding defining the tasks finely enough and scheduling sprints log with accuracy.

The rest of the initial process has been made with respect to the nowadays well-known idea of "less is more". That is to say, we tried to focus on the few practices prescribed by Scrum and only them:

- Fixed time boxes for sprints
- Prioritized backlog
- Sprint planning meetings
- Daily meetings
- Sprint review meetings
- Burn down charts

Consequently, a sprint of ours would be represented as following:

![Figure 11: Initial sprint design](image)

The release plan on the other hand was a bit difficult to forecast at this stage. As a matter of fact, the project met at first a Formetris latent need for generalization more than a precise request of its customer. Which means, a more evasive budget and schedule. The project releases planning was based on granted research and development investment effort. It has then been decided to allocate two months to develop an internal prototype within a first release while Formetris salespeople would look for a customer with an actual project to sustain further development in a second release. In doing so, a consistent base would be proposed internally before working in collaboration with actual end-users and sustainable funding.

![Figure 12: Initial release plan](image)

The Scrum ceremonial being yet defined, we were almost ready to start using it. We knew how to process but not what to process. Hence, we chose to give us up to two weeks to elect a technological solution, gauge the feasibility (Proof Of Concept) and populate our backlog with feature-level tasks (See paragraph Feasibility - Proof of Concept).
4.2.2 Define monitoring tools

To monitor the progression and manage the backlog, we chose to use Greenhopper, an add-on of the JIRA project tracking software (already in use within Formetris). Greenhopper provides dedicated tools for Scrum such as:

- Post-it-like issue tracking organized backlog
- Sprint task board
- Periodical charts generation (such as burn down chart, velocity chart, cumulative data flow...)

Finally, I designed a couple of lightweight templates to help document complex features and perform the sprint retrospections.

4.2.3 Technical choices and gross conception*

As mentioned above, before actually developing a system with the Scrum methodology, it has been decided to perform an early study helping us to populate the backlog and gain visibility.

Available off-the-shelf solutions*

Attentive not to "re-invent the wheel", we started to look for existing software solutions that we could integrate to the evaluation platform. Some of them offered the set of functionalities suiting our ambitions (connectivity with a MySQL database and the possibility of automatically sending mails). However, most of them were Java frameworks that would require important effort to bring them to work together with PHP (interoperability and server technology changes). Further more, the competitive edges of these off-the-self frameworks mostly resided in high level features which are yet not required by our workflows (versioning, graphical visualization, transaction compliancy, very complex rules). In short, while willing not to re-invent the wheel, neither did we want to use a cannon to kill a mosquito! Naturally the choice of implementing our own workflow engine has been made. It is indeed another wheel, but so far a more comfy one.

Home made workflow engine*

Our workflow model consists of questionnaires that can be seen as "states" of a state machine and treatment over their answers which would represent "transitions" linking questionnaires with questionnaires.

The concept of transition actions is actually very similar to the weaving actions described earlier. The challenge was indeed to make these weaved actions reusable from questionnaires to questionnaires instead of questionnaire specific as they were (i.e. hard-coded).

This implies:

- Breaking down the weaving actions into more elementary actions
  - send an email
  - create a user profile
- generate an invitation to a questionnaire
- ...

- Defines for each elementary action a set of parameters to configure its behaviour.
- Define how to pass these parameters and triggers the actions
- Define a mechanism that allows the engine to associate a workflow with questionnaires and questionnaires with a workflow.

Further more the new workflow engine module is explicitly required to be fully compatible with the platform as it is designed. It shall not interfere with linear evaluation processes in use or, more precisely, respect the current way the system stores and retrieves questionnaires and users related data.

Feasibility - Proof of Concept*

A proof of concept is an early transposition of a chosen solution to a specific problem in order to grossly state on its feasibility. In our case, it consisted in modeling, with help of our home made engine "transition actions" the most complex survey workflow that Formetris has designed for its customer: the 360-degree feedback survey. That is to say, define the set of elementary actions that would mimic the behaviour of the hard-coded survey and parameterize them. The 360-degree feedback of an employee consists of a comparison between employee’s subordinates, colleagues, and managers’ feedbacks and a self-evaluation.

The hard-coded workflow then in use was behaving following the pattern below:

1. Evaluated user is invited to the 360-feedback survey*
2. Evaluated user fills a form to invite his co-workers to evaluate him.
3. Weaved post-save action creates evaluators profiles out of form submitted answers and invite them to answer an evaluation questionnaire*
4. Evaluators answers the evaluation questionnaire
5. Evaluated user answers a self-evaluation questionnaire
6. A comparative report is computed and sent*

The stakes were then to breakdown the actions signaled with a "*":

1. The evaluated user profile is created.
2. An invitation for the evaluated to the ’autoevaluation’ questionnaire is generated.
3. The invitation is sent by email.
4. An invitation for the evaluated to the ’invite evaluators’ questionnaire is generated.
5. The invitation is sent by email.
6. The evaluated answers the ’invite evaluators’ with the contact information of the people that will evaluate him.
7. Evaluator profiles are created out of the answers.
8. Invitations for the evaluators to the ’evaluation’ questionnaire are generated.
9. Invitations are sent by email.
10. Once, both the ’evaluation’ questionnaires and the ’autoevaluation’ questionnaire are answered, a comparative report is generated.
11. The report is sent by email.

Additionally we want the questionnaires to be closed at certain dates. As well, we want the reminder mails to stop being sent five days before that closing date which involve handling relative dates (date B is equal date A minus five days). Moreover, if the survey process is suspended for any reason, we want the different questionnaires to closed together. That is to say, if we close/reopen the "autoevaluation" questionnaire, we want the "evaluation" questionnaire to be closed/reopen subsequently. We called this feature "chaining of events".

A graphical version of this process is available in the appendix.

The most complicated treatment of the process is described below. Whether or not it is possible to model it with help of elementary transition actions then represents our proof of concept.

This treatment would:

- Create the evaluator profiles.
- Map the answers of the questionnaire with the evaluator profiles fields (name, surname, email...).
- Create an invitation for each evaluator to answer the "evaluation".
- Send the invitations by mail.
- Set the closing relative dates.
- Initialize the invitation event chainings.

Preliminary study led us to consider a combination between a PHP generic engine and specific XML workflow descriptions. An XML file will be in charge of describing the transition actions (and their parameters) of given process as well as relative dates and event chainings. The PHP engine on the other hand will ensure the exploitation of these descriptions and perform actual data manipulations. The interpretation part of the engine (PHP) is designed to be heavily developed once while the "textual" process descriptions (XML) offer a very light and flexible way of creating and modifying complex survey processes.

Benefits of the PoC

This rather short preliminary phase gave us really valuable inputs. We had now a relative vision of the software components to implement and an approximate idea of the way they would be articulated together. Very precious material came out of the PoC helping us to stack feature-level cards onto our product backlog.

4.3 Initial backlog

The last remaining issue before starting the development with Scrum was to define the initial product backlog. In light of the preliminary study, we then had material to constitute it. But then again, there were many possible ways of doing it. To start with, we chose to use different issue types.

- **Story**
  
  A story is the transcription of an expressed need from the end-user perspective. It is the highest issue level. It is expected to help us keep in mind, at all times, the end purpose of what we are doing.

- **New feature**
  
  A feature is a macro functionality that participate to stories achievement.

- **Task**
A task is the most elementary level of an issue. It represents a precise assignment which results in tangible artifact (e.g., piece of code, design document, runnable test scenario).

- **Improvement**

  An improvement represents a task that does not necessarily increases immediate software value from the end-user perspective. It is rather question of an investment made to reduce later on time-to-market and maintainability costs (e.g., code refactoring).

- **Bug**

  A bug represents additional work required on previously performed task due to quality straws. Important number of bugs is an indicator of process weaknesses and room for betterment.

With respect to this nomenclature, we came out with the initial project backlog presented below.

Scrum encourages frequent face-to-face communication that keep up a shared vision among team members. It is not intended to keep widely understandable traces of development on the long run (as a Software Requirement Specifications document would do). To facilitate reader understanding, I had to slightly modify the issues as they appeared in JIRA. My goal here is not to give a deep comprehension of the technical matter at stake (which I repeat is purely optional) but to stick as much as I can to reality and preserve a certain sense of authenticity.
<table>
<thead>
<tr>
<th>Id</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLF-232</td>
<td>Get familiar with PHP programming and XML parsing mechanisms.</td>
<td>Task</td>
</tr>
<tr>
<td>PLF-231</td>
<td>Implement unit test for XML to PHP objects conversion.</td>
<td>Task</td>
</tr>
<tr>
<td>PLF-220</td>
<td>Define and Implement configuration objects.</td>
<td>Task</td>
</tr>
<tr>
<td>PLF-223</td>
<td>Implement php functions that instantiate configuration objects from an XML file.</td>
<td>Task</td>
</tr>
<tr>
<td>PLF-225</td>
<td>Link a questionnaire to an XML process through the database.</td>
<td>Task</td>
</tr>
<tr>
<td>PLF-217</td>
<td>Implement the &quot;Transition action&quot; function.</td>
<td>New Feature</td>
</tr>
<tr>
<td>PLF-218</td>
<td>Close an invitation after a given date.</td>
<td>New Feature</td>
</tr>
<tr>
<td>PLF-219</td>
<td>Generate a back-office interface to monitor a 360-degree feedback survey.</td>
<td>New Feature</td>
</tr>
<tr>
<td>PLF-228</td>
<td>Implement the entry point of the product process instantiation.</td>
<td>New Feature</td>
</tr>
<tr>
<td>PLF-227</td>
<td>Design and implement the event propagation mechanism over linked invitations.</td>
<td>New Feature</td>
</tr>
<tr>
<td>PLF-190</td>
<td>As a survey customer I want to define my composite survey by answering a questionnaire.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-202</td>
<td>As a back-end operator I want the opening/closure of the components of a composite survey to be automatically chained so I can focus on other tasks.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-199</td>
<td>As a back-end operator I want to easily check the status of the different components of composite surveys.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-194</td>
<td>As a survey customer I want to receive mail alert so I am noticed in real-time if a group of respondent gave, in synthesis of a composite survey, a particular answer to a particular question.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-192</td>
<td>As a survey customer I want to receive the survey report as soon as the last respondent answered the survey.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-198</td>
<td>As a back-end operator I want regenerate a composite survey report so I can re-send it to a survey customer on request.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-197</td>
<td>As a back-end operator I want to activate/deactivate a survey invitations over a composite survey.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-195</td>
<td>As a survey customer I want to define my intended respondents through a CSV file so don’t need to type their contact info manually.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-191</td>
<td>As a survey customer I want to receive regular partial reports.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-201</td>
<td>As a back-end operator I want to deactivate the mail (but leave the invitation open) on request.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-200</td>
<td>As a back-end operator I want to re-send a mail invitation on request.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-203</td>
<td>As a survey respondent I want to receive mail reminders so I don’t forget to answer a questionnaire.</td>
<td>Story</td>
</tr>
<tr>
<td>PLF-230</td>
<td>As Formetris contractors I want to offer to my customer workflow compliant survey tools.</td>
<td>Story</td>
</tr>
</tbody>
</table>

One has got to start somewhere

So here we are. We have:

- The "Why": Formetris needs for flexibility.
- The "Who": Yann and I (possibly joined by Hassan and Yinjia on further customer releases).
- A certain idea of "When": A first five-month release.
- A certain idea of "What": An initial product backlog.
- And a certain idea of "How": Our Scrum ceremonial.
At the end of this first phase of reflection, I was convinced that no matter how long I could keep thinking over it sat at my desk, there was no way to design an optimal Scrum process running from the start out of the information I disposed of. I was well aware that the particular combination of choices we had made, like every other combination that we could have came up with, was wrong! The whole point is how wrong it was. I keep seeing Scrum as the Newton’s method to find a function’s roots. Most start points would eventually lead to a satisfying solution. However, the choice of that point is crucial. The closer it initially is from the solution, the faster the method converges. With help of the preliminary process design phase we intended to spare us most of the common pitfalls and hopefully fasten the adoption of a truly beneficial Scrum process.

The chart above illustrates the balance that we tried to tilt in our favour. Instead of starting working in a hurry with a potentially poor efficiency at first (process B), we took time to reflect expecting that the delay caused by it would be paid back (hopefully with interest!) by a higher productivity (process B). Of course, the shorter the project is, the most importance it might have. On the long run, both processes are likely to reach satisfying productivity.

5 Early steps sprints’ retrospection

As we could expect the process was, at first, relatively erratic. Sprint retrospections only took place after sprint 2 and 4 instead of every week as planned. The first sprint did not provide us with enough material to perform a retrospection, the time tracking has not been very rigorous and a misuse of JIRA led to inconsistent burn down chart. Sprint number 3 was disturbed by heavy snow falls preventing some team members to go to work for several days. Though, retrospections of sprints 2, 4, 5 and 6 (which you can find in the appendix as well as their sprint logs) gave use valuable inputs.

Sprints 1 and 2 stated the following strong points:
• One-week sprints seem to be accurate (allows for great reactivity against changes as we go).
• The pre-study gave precious input for task prioritization.
• Education plan goes smoothly and PHP(Zend) skills increase towards meeting actual needs.
• Test-driven development on logic critical functions gave great results.

And the following areas for improvement:
• Time tracking needed to be done more rigorously.
• Some tasks were underestimated or ignored (unit test, integration, planning, refactoring).
• Tasks in sprint log were too big and overlapping (thus difficult to estimate).
• Sprints lacked of slack time.

Out of these insight, we decided to emphasize on:
• breaking down the tasks more finely to improve estimate accuracy.
• dedicating 15-20 minutes at the end of the day to properly log work and commit code on versioning system.
• encouraging the use of TDD on critical parts.

Sprints 3 and 4 stated the following strong points:
• Time tracking has been done more rigorously.
• Code refactoring effort and compliance to coding rules facilitated debugging and integration.
• Peer programming for integrating separately developed module has been significantly efficient.

and the following areas for improvement:
• Maintenance of the unit tests has been skipped.
• Integration time has been way more complicated than expected (underestimated).

Finally, sprints 5 and 6 highlighted the following items:
• “Just in time architecture”, TDD and regular code reviews have a great effect on keeping high software quality standards and a valuable potentially shippable product.

and the following areas for improvement:
• Decreasing quality of sprint log definition (lacks of slack time, coarse-grained tasks, mispriorization).

At the end of this first period within which the efforts made were focused on stabilizing and improving the flow of the initial process, we decided to identify its flaws and weaknesses and introduce new pratices to improve the process itself.

Part V
Incorporation of scrum-compatible practices

After the sixth sprint, important changes occured in the project. Yann, our PO, was two weeks off and Formetris negociated the final terms of a contract with a customer. We observed a period of transition during which I could prepare the upcoming release. Based on the new customer needs, the next release’s scope was now to implement a bottom-up feedback survey. The bottom-up feedback differs from the 360-degree feedback in that managers are only evaluated by their subordinates instead of their subordinates, colleagues and superiors.

It is here important to note that the project nature identified earlier (see paragraph 4.2.1 - Define Scrum ceremonial) has been altered. It has now:
• A very detailed level of specification.
• Immediate applicative purpose.
• A much less available customer.
• A precise release duration (5 weeks).
• Higher understanding of project outlines and task dependencies.

This transition was a valuable occasion to perform a release retrospection. In order to do so, I chose to synthetize the different sprints’ retrospectives mentioned earlier and drawn from it the following items:

• Strengths
  – Communication and impediment removal
  – Mastery of in-use technologies
  – Conservation of high software quality standards

• Areas of development
  – Accuracy of task estimates
  – Compliance with schedule
  – Task breakdown

Looking at it, we could state on the relatively good quality of our scrum ceremonial while identifying needs for additional monitoring tools concerning scheduling and estimation. The next paragraphs then describe the reflection and the actions undertaken to correct these particular points.

6 Incorporation of velocity

During the first release, velocity charts were not formally used to perform the retrospections. According to the initial design of our process, the only burn down charts were used to monitor progression. However, the burn down chart gives a very punctual snapshot of progression that helps regulate the pace within a sprint. It gave us important feedback about the likeliness of completing tasks before the end of a sprint or identify the particular tasks (or kind of tasks) that cause difficulties or are regularly underestimated. Nevertheless, retrospections require a bigger picture of progression that would help regulate the pace over the sprints.

Although we had not been using the velocity chart so far, it has been automatically computed by Greenhopper. As we can retrospectively see on them, sprints increased in productivity during this early phase (sprints 1, 2 and 4).
As the absolute amount of performed work increased, the difference between expected and actual work got closer and closer (decreasing error) and the forecasted slack time decreased. This trend may be explained an increased mastery over the technologies in-use which allowed me to perform more valuable tasks in a consistent amount of time and a better understanding of mutual work capacity between Yann and I (that led to a more accurate time estimate).

There is however no room for complacency, one should not loose focus on continuously improving the process nor on stressing the use of positive practices. As a matter of facts, the next two sprints were scheduled less carefully. We lost sight on controlled progression and drifted toward a "let’s do as much as we can during this sprint” kind of schedule. This respite gave as a result: unrealistic commitment, drastic lack of slack time, high error and fewer accomplished work.
Further more, these unrigorous sprints left us with a twisted visibility for upcoming sprints and an erratic software quality control. At the end of the sprint, it was hard to distinguish what has been completely done from what was left partially uncompleted and if make harder as well the estimation of the upcoming tasks.

Speaking with figures, the past sprints provided us with the following data:

<table>
<thead>
<tr>
<th>Sprint</th>
<th>Commitment (h)</th>
<th>Completed (h)</th>
<th>Slack time (h)</th>
<th>Slack time (%)</th>
<th>Error (h)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint 1</td>
<td>20</td>
<td>13</td>
<td>15</td>
<td>43%</td>
<td>7</td>
<td>35%</td>
</tr>
<tr>
<td>Sprint 2</td>
<td>23</td>
<td>21</td>
<td>12</td>
<td>34%</td>
<td>2</td>
<td>9%</td>
</tr>
<tr>
<td>Sprint 4</td>
<td>27</td>
<td>27</td>
<td>8</td>
<td>23%</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Sprint 5</td>
<td>30,5</td>
<td>20</td>
<td>4,5</td>
<td>13%</td>
<td>11</td>
<td>34%</td>
</tr>
<tr>
<td>Sprint 6</td>
<td>27,5</td>
<td>19</td>
<td>7,5</td>
<td>21%</td>
<td>9</td>
<td>31%</td>
</tr>
</tbody>
</table>

The data set being rather small and the variability of project factors high (growing skills, mutual understanding, project nature), it was not possible to extract generic long-run trends. Nevertheless, it appeared important to our eyes to observe a minimum rate of slack time per sprint that we initially and arbitrarily fixed to 30%.

In the future sprints, the incorporation of the velocity chart analysis is expected to provide us with over-sprints trends. These trends would eventually help us to calibrate the effort on base of team capacity and hopefully reduce the amount of uncompleted tasks and associated setbacks [11][6][4].

7 Incorporation of planning poker

Forcing us to forecast an amount of slack time above a certain limit indeed allowed us to regulate the sprints and reduce significatively the number of uncompleted tasks. However, the retrospectives following its introduction showed that sprints schedules were occasionally pessimistic and the slack time filled with unscheduled tasks (see Annex 20 - Sprint 7 & 8 - Retrospection). Though it is more confortable, in terms of
customer-relationship, to respect a pessimistic delivery date than hurry up to catch up an optimist one, our very purpose here is to know accurately our time-to-market. Therefore, after having improved our process on a macro-level by introducing a minimal amount of slack-time based on an over-cycles observations, we aim now an improvement on a more atomic-level (single tasks). As a matter of fact, the deviation between forcasted, committed and achieved work over a sprint results from the addition of deviations between forcasted, committed and achieved work over single tasks. Setting a minimum slack-time limit per sprint thus only constitutes a safeguard and a limited lever of improvement.

We decided then to introduce the so-called "planning poker"[5] in our sprint planning meetings.

During the sprint planning meeting, each member of the team has numbered cards in his or her hands. When a task is elected by the PO, each of the members, independently one from another, calls the card on which figures, in his or her opinion, the correct estimate of the task. The members who called the lowest and highest cards are invited to justify their estimates and discuss it until they hopefully find a consensus about the estimate.

![Figure 17: A scrum team using planning poker](image)

The benefits which can be expected of using planning poker are well known[13]. It is likely to reduce the bias when performing estimation. As a matter of facts, PO and team members, cannot simply estimate tasks with objectivity. They have different positions to hold[14] and, more or less conscienciously, base their estimates on these. The estimate which is suppose to be a forecasted duration of the time actually needed to achieve a task indeed becomes a wished delivery date. The POs traditionally tend to underestimate (the sooner the better) and the team to overestimate (the later the better). However, the team tends to be influenced by the first expressed estimate[14]. When a team members innocently claim "this task is critical and requires our full attention, let’s grant it 25 hours", the PO will express his opinion on the estimate by altering this very estimate. If he thinks it is too few he migh say 27 or 23 if too much but only rarely contest it entirely. On the other if the PO says "This task is really simple, we should get this done in half day", the team will very unlikely express an estimate far from it. When both PO and team members are expressing an estimate at the very same time, a real unbiased debat can take place and hopefully the subjectivities neutralize one another.

In our case, more than neutralizing the bias, the very beneficial effect observed while using planning poker is an out-loud formalization of our estimates. Indeed, instead of roughly picking up a plausible number when estimating a task, the planning poker forces us to construct a mental representation of the atomic subtasks which constitute this task and objectively discuss about it[9].

In addition to that, the planning poker helped us to point out certain tasks that tended to be regularly underestimated such as graphical finishes, writting code comments, writing unit tests, overestimated as refactoring or forgotten[9].

8 Sprint reviews & retrospections

Using the data brought by the lastest sprints’ retrospections (which can be found in the appendix as well as their sprints’ logs) it is possible to draw the following velocit chart:
Leaving aside the 9th sprint which, due to project delivery and the very frequent back and forth with the customer to finalize the details, has been quite erratic, a general improvement trend can be observed. While keeping a rather constant amount of slack-time, the amount of achieved work (and consequently the amount of work which we are invited to commit to) increased. After the introduction of velocity and planning poker, the error between committed and achieved work has reached a maximum level of 7% compared to an average of 27% (and a maximum of 35%) earlier. Beyond this overall performance betterment of our process, some sprints, especially those immediately following the introduction of a minimum slack-time limit, witnessed the apparition of new deficiencies:

- Some sprints were pessimistically scheduled and the team was brought to work on unscheduled tasks to complement it.
- Though the delivery went well, the delivery sprint slept out from control regarding scheduling and tracking aspects.
- Punctually, the code produced was not of "good enough" quality (too rarely commented, un-adjusted copy/paste from code, frequent style-check errors).

Moreover, some of the flaws identified within retrospections have not been corrected:

- Retrospections on busy sprints tend to be skipped.
- Tasks, eventhough informally detailed while using planning poker, are not broken down clearly enough in sprint log.

Sprint number 14, which constitutes the last sprint before August code-freeze, marks the end of this thesis work’s scope and concludes our study.
Part VI  
Recommendations and Conclusion

9 Qualitative observations

At the end of this second release, the process feels to be better controlled and the team significantly more self-aware of its capacity and limits. The decisions now made at the different steps of the process (planning, development and retrospection) are based on constructive discussions and tangible indicators. Whenever sprints do not meet expectations, it is not anymore a complete surprise but rather the result of liberties deliberately taken during sprint planning when the context allows or requires it (e.g. delivery sprint, pre-"code-freeze" sprint). Furthermore, the main difficulty that we encountered, the poor accuracy of tasks estimation, seems to have reached a satisfying balance for the time being with help of velocity and planning poker practices.

Apart these positive outcomes and the "good-enough" quality of software produced, there is no room for triumphalism. The freshly introduced practices are to be observed on a longer-run and the rigour of their use tirelessly recalled. Code quality as well is to be closely monitored as it has punctually been degraded.

To conclude this paper, I would like to retrace humbly was I consider to be the strong points of this early Scrum implementation before opening a little parenthesis about the possible action to be undertaken in the next few sprints.

10 What has worked out

As mentioned in the first paragraphs, the main difficulty encountered when one is about to use Scrum is to fill the significant process-gap(deliberately) left undefined by the framework. Sprint duration, monitoring, quality assessment, initial backlog elicitation, team capacity forecast are few examples of what the new practitioners need to define and for which literature is of poor help. Beyond the question of taking good or bad decisions, the most important is to objectify them in order to gain confidence and motivation to leap into the unknown. We made it by qualitatively identify the distance that could exists between our project and a supposed ideal project (i.e. which would be compatible with Scrum ideal prescriptions). Each deviation from this model had consequences on the decision we made (e.g. highly unstable architecture causes reduced team capacity). Though it is difficult to judge if the process we came up with maximized the likeliness to succeed, the fact if was not simply based on gut feelings brough us confidence and positively participated in Scrum implementation.

Another destabilizating factor is the consistent number of "best practices" one can choose from to customize the process. Most of the time these practices are presented in the literature as being so highly positive for software quality assessment and process monitoring that one ends up to see the simple Scrum process without them pointless and doomed to failure. If you are to buy a car, you buy the one that has got as much options as you can afford not the entry-level one. However if you are a rookie in auto mechanics and your car disfunctions, you would easier repair the entry-level one. Therefore, the process designed, being most likely to disfunctions at first, we decided to keep it as simple as possible with as few "optional" practices as possible. In doing so, process flaws are immediately identifiable and the corresponding practices to address them introduced when needs arises. Hence we started using Scrum with very few practices, used test driven development and quick desing session only when needed and introduced velocity and planning poker on a regular base when recurrent estimation issues were verified.

With helps of the two heuristics presented above the initial process-oriented doubts were seemingly removed: the "How" was defined. The "What" however remained uncertain. Though projects outlines were clear, planning a sprint was hard. Without having a gross idea of the solution to implement, e.g the technology to use, we had no clue of how to define, prioritize and estimate tasks. Would "explore and elect a technology" be a suitable task to be scheduled in a sprint? We do not have answers to that, however the choice was made to carry out a pre-study apart from Scrum scope to give the implementation an initial
direction and elect a bunch of technical coarse-grained tasks to start with.

Not surprisingly, once our release kicked off, the main support to carry on in the right direction was found in performing retrospections regularly. To be fair, retrospection is actually recommended unanimously in the literature and it is thus no big discovery to write here that it is essential. Nonetheless, just as we experienced a couple of times, when schedule gets loaded it is what get neglected first. This is most probably because it does not give the impression of immediately producing software. As far as I am concerned, I consider it to be, on the long run, the most important source of added value. Without it indeed, the team caught up by its own rhythm never takes time to introspect its work and does not have a chance to identify and address flaws. Sprints schedule is always overan? Is it communication issues? Estimation inaccuracy? Technical debt cost? There is no way to find, nor work toward, a solution. How can someone reach a goal destination without knowing where he is and which direction he is going?

Among the last recommendation that I want to share here would be, ”give yourself time when introducing changes or new practices in the process” and ”one change at a time”. According to our own experience, the beneficial effect of changes in the process, if any, appears progressively. Because of inexperience and/or change reluctancy the true potential of a tool is revealed as its use spreads out in time. Introducing them separately over several sprints allows to identify their inherent effects.

11 For the next sprints

Over sprint estimation and planning aspects have been stabilized to a performant enought level that allows good control over progression and deadline respect. In my opinion though, they still consititute the main room for improvement especially regarding task breakdown, estimation justification and re-use of past estimates on similar tasks which figure among identified valuable factors of estimation accuracy [9]. Therefore, after observing the effects of velocity and planning poker on a longer-run, if such improvement potential still exists we might use:

- Maximal task size (e.g. 2-3 hours) [16].
- Incorporate Kanban to Scrum (Scrumban) to formalize how the team should react to suddent high priority task[10][7] (e.g. bug fix on delivered version).
- Incorporate an Evidence-Based-Scheduling[16] tool that calculates estimates probalility based on sim-ilar past tasks.
- Separate tasks estimation process from sprint scheduling process to avoid estimating tasks conveniently for them to fit the sprint[9].
- Combine different of the above methods to benefit from triangulation.

Coming right after schedule and planning aspects, figure code quality assessment. In the present process we tend to product working software in one sprint, and most of the time, review, refactor and/or clean it in the following. One of the drawback observed in doing so, is the negligence of theses ”adjustment” tasks when the schedule gets busy. To avoid it, we might consider establishing a ”Definition of Done”[15] that would define a set of criteria without which a task cannot be considered as ”Done”. For example we could include in this DoD: compliancy with style checker, impose a certain ratio of comment line per code lines and a minimum unit test code coverage. We would thus be encouraged to perform these ”adjustment task” in the same sprint as code production and allocate it the time and attention it requires (possibly by formally dedicate the friday afternoon for such code adjustments).

Finally, we could limit the number of impediments to be removed from on retrospection to the other and force one impediment to be removed before considering others to prevent from just piling them up but actuallty working toward their removal[2][10].
References


Part
Annexes

1 The 360-degree feedback process
Create evaluator’s invitations to questionnaire “evaluation”

Send evaluator’s invitation

AND

Generate report

Send report

End

Evaluators answer to questionnaire “evaluation”

Evaluated

Evaluators

System

Etienne Girot

Spring 2013
## 2 Sprint 1 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Issue Type</th>
<th>Status</th>
<th>Original Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLF-220 *</td>
<td>Define and Implement configuration objects.</td>
<td>Task</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>PLF-232 *</td>
<td>Get familiar with PHP programming and XML parsing mechanism.</td>
<td>Task</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>PLF-223 *</td>
<td>Implement php functions that instanciate configuration objects from an XML file.</td>
<td>Task</td>
<td>Completed</td>
<td>6h</td>
</tr>
<tr>
<td>PLF-231 *</td>
<td>Implement unit test for xml to object conversion</td>
<td>Task</td>
<td>Uncompleted</td>
<td>7h</td>
</tr>
<tr>
<td>PLF-216 *</td>
<td>Parse XML configuration file and convert it into configuration objects.</td>
<td>New Feature</td>
<td>Uncompleted</td>
<td>-</td>
</tr>
</tbody>
</table>
# 3 Sprint 2 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Issue Type</th>
<th>Status</th>
<th>Original Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLF-234</td>
<td>Define and implement a dedicated Registry for the configuration loader.</td>
<td>Improv.</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>PLF-235</td>
<td>Use the Zend_Date object to store and execute operation on dates and add a fromXML() functions for dates using it.</td>
<td>Improv.</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>PLF-237</td>
<td>Implement unit test that will ensure that for a given XML configuration, the right entries are inserted in the database.</td>
<td>Task</td>
<td>Completed</td>
<td>7h</td>
</tr>
<tr>
<td>PLF-231</td>
<td>Implement unit test for xml to object conversion</td>
<td>Task</td>
<td>Uncompleted</td>
<td>2h</td>
</tr>
<tr>
<td>PLF-233</td>
<td>Fix the configuration feature to be automatically loaded by Zend without specifying the &quot;included&quot; PHP files.</td>
<td>Bug</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>PLF-225</td>
<td>Link the function with a questionnaire through the database.</td>
<td>Task</td>
<td>Uncompleted</td>
<td>-</td>
</tr>
<tr>
<td>PLF-217</td>
<td>Implement the &quot;GenerateInvitation&quot; 360° post-treatment</td>
<td>New Feature</td>
<td>Uncompleted</td>
<td>-</td>
</tr>
<tr>
<td>PLF-216</td>
<td>Parse XML product configuration file and convert it into configuration objects.</td>
<td>New Feature</td>
<td>Uncompleted</td>
<td>-</td>
</tr>
</tbody>
</table>
4 Sprint 3 - Planning

Not available.
## 5 Sprint 4 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Type</th>
<th>Status</th>
<th>Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLF-249</td>
<td>Send invitation mail instantaneously after the ListEvaluator questionnaire has been answered.</td>
<td>New Feature</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>PLF-246</td>
<td>Settle an actual demonstration of the system over a 360-degree survey.</td>
<td>Task</td>
<td>Completed</td>
<td>6h</td>
</tr>
<tr>
<td>PLF-244</td>
<td>Debugging</td>
<td>Bug</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>PLF-244</td>
<td>Refactor code</td>
<td>Improv.</td>
<td>Completed</td>
<td>2.5h</td>
</tr>
<tr>
<td>PLF-242</td>
<td>Refactor unit testing</td>
<td>Improv.</td>
<td>Completed</td>
<td>2.5h</td>
</tr>
<tr>
<td>PLF-251</td>
<td>Modify AnswerCollection-find/save/fetch function to handle passed-by-reference Answers.</td>
<td>Task</td>
<td>Completed</td>
<td>5h</td>
</tr>
<tr>
<td>PLF-250</td>
<td>Modify DB table schema to incorporate the question reference mechanism (add association table AnswerCollection_AnswerCollection_Answer).</td>
<td>Task</td>
<td>Completed</td>
<td>1h</td>
</tr>
<tr>
<td>PLF-217</td>
<td>Implement the &quot;GenerateInvitation&quot; post-treatment</td>
<td>New Feature</td>
<td>Completed</td>
<td>-</td>
</tr>
<tr>
<td>PLF-252</td>
<td>Modify InvitationConfiguration-execute to pass by reference answers when necessary</td>
<td>Task</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>PLF-248*</td>
<td>Create a XSD to validate workflow XML file and implement validation mechanism in platform.</td>
<td>Improv.</td>
<td>Uncompleted</td>
<td>-</td>
</tr>
<tr>
<td>PLF-228</td>
<td>Implement the entry point of the product process instantiation.</td>
<td>New Feature</td>
<td>Uncompleted</td>
<td>-</td>
</tr>
</tbody>
</table>
## 6 Sprint 5 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Issue Type</th>
<th>Status</th>
<th>Original Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLF-257</td>
<td>Encapsulate mapping arrays into a dedicated class.</td>
<td>Improv.</td>
<td>Completed</td>
<td>1.5h</td>
</tr>
<tr>
<td>PLF-258</td>
<td>Write retrospection for sprint 4</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>PLF-259</td>
<td>Establish a list of potential tools for Scrum, ScrumBan, EvidenceBasedScheduling ...</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>PLF-262</td>
<td>Populate the RelativeDate table from XML.</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>PLF-261</td>
<td>Create DB scheme for RelativeDates handling</td>
<td>Task</td>
<td>Completed</td>
<td>1.5h</td>
</tr>
<tr>
<td>PLF-264</td>
<td>Set closing date in demo for today and verify that questionnaire is not editable.</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>PLF-254 *</td>
<td>Create a TransitionEmail transition action.</td>
<td>Task</td>
<td>Completed</td>
<td>5h</td>
</tr>
<tr>
<td>PLF-263</td>
<td>Retrieve dynamically-calculated date.</td>
<td>Task</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>PLF-260</td>
<td>Document the xml schema and domain specific language interpreter</td>
<td>Task</td>
<td>Completed</td>
<td>3.5h</td>
</tr>
<tr>
<td>PLF-266</td>
<td>Add a step to the demo. To close all questionnaire 3 days before trainee session date (relative dates).</td>
<td>Task</td>
<td>Uncompleted</td>
<td>7h</td>
</tr>
<tr>
<td>PLF-218</td>
<td>Close an invitation after a given date.</td>
<td>New Feature</td>
<td>Uncompleted</td>
<td>-</td>
</tr>
</tbody>
</table>
# 7 Sprint 6 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Issue Type</th>
<th>Status</th>
<th>Original Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLF-269</td>
<td>Implement SQL requests for fetching all invitation which invitation closing date are ”today” (using relative date : consider using PL-SQL).</td>
<td>Task</td>
<td>Completed</td>
<td>5h</td>
</tr>
<tr>
<td>PLF-268</td>
<td>Think about and sketch back-office look and feels for monitoring workflows.</td>
<td>New Feature</td>
<td>Completed</td>
<td>5h</td>
</tr>
<tr>
<td>PLF-266</td>
<td>Add a step to the demo. To close all questionnaire 3 days before trainee session date (relative dates).</td>
<td>Task</td>
<td>Completed</td>
<td>7h</td>
</tr>
<tr>
<td>PLF-265</td>
<td>Refactor SQL script to remove ’autoevaluation questionnaire’ and display role in stakeholders list.</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>PLF-218</td>
<td>Close an invitation after a given date.</td>
<td>New Feature</td>
<td>Completed</td>
<td>-</td>
</tr>
<tr>
<td>PLF-270</td>
<td>Try to implement tiggers for denormalization.</td>
<td>Task</td>
<td>Uncompleted</td>
<td>5h</td>
</tr>
<tr>
<td>PLF-260</td>
<td>Document the xml schema and nested code interpreter</td>
<td>Task</td>
<td>Uncompleted</td>
<td>3.5h</td>
</tr>
</tbody>
</table>
8 Sprint 7 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Issue Type</th>
<th>Status</th>
<th>Original Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEA-28</td>
<td>Implement SQL script for questionnaire &quot;evaluation&quot;</td>
<td>Task</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>BEA-29</td>
<td>Implement SQL script for questionnaire &quot;autoevaluation&quot;</td>
<td>Task</td>
<td>Completed</td>
<td>3.5h</td>
</tr>
<tr>
<td>BEA-30</td>
<td>Implement script for questionnaire &quot;add wished evaluator&quot;</td>
<td>Task</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>BEA-31</td>
<td>Implement SQL script for questionnaire &quot;add manager to evaluate&quot;</td>
<td>Task</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>BEA-35</td>
<td>Adjust CSS to display nicely the questionnaire and send it to customer.</td>
<td>Improv.</td>
<td>Completed</td>
<td>5h</td>
</tr>
</tbody>
</table>
# 9 Sprint 8 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Issue Type</th>
<th>Status</th>
<th>Original Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEA-40</td>
<td>Implement prequestionnaire for evaluators (generate links + views).</td>
<td>Task</td>
<td>Completed</td>
<td>5h</td>
</tr>
<tr>
<td>BEA-38</td>
<td>Implemente SQL queries for prequestionnaires.</td>
<td>Task</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>BEA-39</td>
<td>Implement prequestionnaire for managers (generate links + view)</td>
<td>Task</td>
<td>Completed</td>
<td>5h</td>
</tr>
<tr>
<td>BEA-36</td>
<td>Finish implementing the &quot;add wished evaluator&quot; post-save function. (+ test and debug)</td>
<td>Task</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>BEA-37</td>
<td>Implement the &quot;Add manager to evaluate&quot; post-save function.</td>
<td>Task</td>
<td>Completed</td>
<td>5h</td>
</tr>
</tbody>
</table>
10 Sprint 9 - Planning

Not available.
# 11 Sprint 10 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Issue Type</th>
<th>Status</th>
<th>Original Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEA-60</td>
<td>Refactor code according to code review.</td>
<td>Improv.</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>BEA-55</td>
<td>Consolidated report - criteria selection page + CSS</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>BEA-54</td>
<td>Real time report - Implement HTML view + adjust CSS.</td>
<td>Task</td>
<td>Completed</td>
<td>0.5h</td>
</tr>
<tr>
<td>BEA-53</td>
<td>Real time report - Build SQL queries to retrieve data to be displayed.</td>
<td>Task</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>BEA-52</td>
<td>Real time report - Set up report in SQL script</td>
<td>Task</td>
<td>Completed</td>
<td>0.5h</td>
</tr>
<tr>
<td>BEA-56</td>
<td>Consolidated report - Parameter transmission between two report pages.</td>
<td>Task</td>
<td>Completed</td>
<td>1h</td>
</tr>
<tr>
<td>BEA-57</td>
<td>Consolidated report - SQL requests + individual report adaptation.</td>
<td>Task</td>
<td>Uncompleted</td>
<td>-</td>
</tr>
</tbody>
</table>
## 12 Sprint 11 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Issue Type</th>
<th>Status</th>
<th>Original Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEA-64</td>
<td>Matrix report - modify the criteria selection page to display the select box to choose scatter plot axes.</td>
<td>Task</td>
<td>Completed</td>
<td>1h</td>
</tr>
<tr>
<td>BEA-63</td>
<td>Matrix report - create the page that displays the matrix.</td>
<td>Task</td>
<td>Completed</td>
<td>1h</td>
</tr>
<tr>
<td>BEA-65</td>
<td>Refactor Datamanager to isolate specific function of the various reports and create and abstract class to gather the common ones.</td>
<td>Improv.</td>
<td>Completed</td>
<td>7h</td>
</tr>
<tr>
<td>BEA-57</td>
<td>Consolidated report - SQL requests + individual report adaptation.</td>
<td>Task</td>
<td>Completed</td>
<td>7h</td>
</tr>
<tr>
<td>BEA-58</td>
<td>Matrix report - Implement a php page that generate javascript code to display a matrix chart (scatter plot).</td>
<td>Task</td>
<td>Completed</td>
<td>5h</td>
</tr>
<tr>
<td>BEA-66</td>
<td>Review CSS, fix remaining details (in particular for criteria selection page).</td>
<td>Improv.</td>
<td>Completed</td>
<td>3h</td>
</tr>
</tbody>
</table>
## 13 Sprint 12 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Issue Type</th>
<th>Status</th>
<th>Original Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEA-69</td>
<td>Model: implement generic real-time stat &quot;number of invitations&quot;</td>
<td>Task</td>
<td>Completed</td>
<td>0.5h</td>
</tr>
<tr>
<td>BEA-72</td>
<td>Model: implement generic real-time stat &quot;number of answers&quot;</td>
<td>Task</td>
<td>Completed</td>
<td>0.5h</td>
</tr>
<tr>
<td>BEA-70</td>
<td>Model: implement generic real-time stat &quot;number of respondents&quot;</td>
<td>Task</td>
<td>Completed</td>
<td>0.5h</td>
</tr>
<tr>
<td>BEA-78</td>
<td>Model: implement generic real-time stat &quot;number of active respondents&quot;</td>
<td>Task</td>
<td>Completed</td>
<td>0.5h</td>
</tr>
<tr>
<td>BEA-71</td>
<td>Model: compute multiple stats from batch configuration object</td>
<td>Task</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>BEA-74</td>
<td>Mediator: retrieve possible choices to a question</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>BEA-73</td>
<td>Mediator: display choices from question in select box with a custom label</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>BEA-76</td>
<td>Mediator: link select boxes to filter several generic stats</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>BEA-77</td>
<td>Test Model</td>
<td>Task</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>BEA-79</td>
<td>Test Mediator</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>BEA-75</td>
<td>Refactor</td>
<td>Improv.</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>BEA-80</td>
<td>Mediator: adjust CSS</td>
<td>Improv.</td>
<td>Uncompleted</td>
<td>1.5h</td>
</tr>
</tbody>
</table>
### 14 Sprint 13 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Issue Type</th>
<th>Status</th>
<th>Original Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEA-80</td>
<td>Mediator: adjust CSS</td>
<td>Improv.</td>
<td>Completed</td>
<td>1.5h</td>
</tr>
<tr>
<td>BEA-82</td>
<td>Prototype: Use the generic stat on a 360-degree survey</td>
<td>Task</td>
<td>Completed</td>
<td>1.5h</td>
</tr>
<tr>
<td>BEA-84</td>
<td>Model: implement individual stats (avg by questions, avg by part)</td>
<td>Task</td>
<td>Completed</td>
<td>5h</td>
</tr>
<tr>
<td>BEA-83</td>
<td>Adjust CSS on report page</td>
<td>Task</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>BEA-81</td>
<td>Adjust CSS when printing report page</td>
<td>Task</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>BEA-85</td>
<td>View: Extend the chart renderer to handle generic stats functionalities</td>
<td>Improv.</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>BEA-87</td>
<td>Model-View: Implement a Dataprocessor which format generic stat into chart compatible data</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>BEA-86</td>
<td>Refactor</td>
<td>Improv.</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>BEA-88</td>
<td>Code review</td>
<td>Improv.</td>
<td>Uncompleted</td>
<td>0.5h</td>
</tr>
</tbody>
</table>
# Sprint 14 - Planning

<table>
<thead>
<tr>
<th>Key</th>
<th>Summary</th>
<th>Issue Type</th>
<th>Status</th>
<th>Original Time Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEA-88</td>
<td>Code review</td>
<td>Improv.</td>
<td>Completed</td>
<td>0.5h</td>
</tr>
<tr>
<td>BEA-94</td>
<td>Write documentation on generic stat generation</td>
<td>Task</td>
<td>Completed</td>
<td>2h</td>
</tr>
<tr>
<td>BEA-92</td>
<td>Handle the &quot;N/A&quot; answers in stats and charts.</td>
<td>Improv.</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>BEA-93</td>
<td>Implement project with on-the-fly invitations (no mail sent, user is unknown).</td>
<td>Task</td>
<td>Completed</td>
<td>4.5h</td>
</tr>
<tr>
<td>BEA-97</td>
<td>Add a &quot;remember my answers and finish later&quot; when answering a survey</td>
<td>Task</td>
<td>Completed</td>
<td>4h</td>
</tr>
<tr>
<td>BEA-91</td>
<td>Refactor consultant questionnaire with on-the-fly invitations</td>
<td>Task</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>BEA-95</td>
<td>Massive user import from lists.</td>
<td>Task</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>BEA-96</td>
<td>Refactor</td>
<td>Improv.</td>
<td>Completed</td>
<td>3h</td>
</tr>
<tr>
<td>BEA-98</td>
<td>Clean code</td>
<td>Improv.</td>
<td>Completed</td>
<td>1h</td>
</tr>
</tbody>
</table>
Sprint retrospection

Sprint #1,2

Abstract
This document describes the positive and negative practices performed during past sprints and define an action plans to improve the process.

Version
<table>
<thead>
<tr>
<th>Version</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialization</td>
<td>E. Girot</td>
</tr>
</tbody>
</table>

Highlighted practices
<table>
<thead>
<tr>
<th>List of factor that worked really well during the sprint just completed.</th>
<th>List of factors that were problematic during the sprint just completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-week sprints seem to be accurate.</td>
<td>Some stories were too big and thus difficult to estimate.</td>
</tr>
<tr>
<td>Daily scrums help to accord requirements with realization efficiently.</td>
<td>Time has not been tracked well enough (overlapping tasks, lacks of rigour). Charts are not helpful.</td>
</tr>
<tr>
<td>Test driven development is very efficient.</td>
<td>Sprint lacked of slack time 34.5 hours booked over 35 available.</td>
</tr>
<tr>
<td>Time spent on modeling the processes was accurate (not too nor too few detailed).</td>
<td>Some tasks were underestimated (unit test, debugging, planning).</td>
</tr>
<tr>
<td>The provided expertise concerning the existing plateform and used technologies (PHP, SQL) allows for efficient education and progression.</td>
<td>Early hesitation regarding software architecture and coding rules led to heavy needs for refactoring.</td>
</tr>
</tbody>
</table>

Actions plan
- Stories are going to be broken down into more atomic tasks. Time estimation is expected to be more accurate and traceability increased since tasks should no longer overlap.
- Working time is going to be logged more rigorously. A daily quarter of hour before leaving the office will be dedicated to logging time in JIRA and committing work on SVN.
- Sprint will involve 25% of slack time to handle uncertainty.
- Daily scrums and increasing experience should improve the knowledge of the existing platform as well as mastery over the used technologies and reduce the need for heavy refactoring accordingly. However, the project at hand being new and at an early stage of its development, substantial refactoring is to be regularly forecasted.
- Positive practices will be continued and encouraged (Test driven development, daily scrums, one-week sprints).
Sprint 3 & 4 - Retrospection

Abstract
This document describes the positive and negative practices performed during past sprints and define an action plans to improve the process.

Version

<table>
<thead>
<tr>
<th>Version</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialization</td>
<td>E. Girot</td>
</tr>
</tbody>
</table>

Highlighted practices

<table>
<thead>
<tr>
<th>List of factor that worked really well during the sprint just completed.</th>
<th>List of factors that were problematic during the sprint just completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair debugging has been significantly efficient.</td>
<td>Integration of components was more difficult than expected and underestimated.</td>
</tr>
<tr>
<td>Time tracking has been performed rigorously.</td>
<td>Some tasks have not been completed.</td>
</tr>
<tr>
<td>Code refactoring and respect of coding rules allowed clear understanding of code and facilitated debugging.</td>
<td>Time needed to refactor unit test according to changes in code after component integration has not been forecasted.</td>
</tr>
</tbody>
</table>

Actions plan

- More time will be forecasted to integrate and stabilize code.
- Implementation tasks time estimation will include test refactoring and code cleaning and will be thus increased. This measure should avoid rush before committing code and increase code stability.
- Working practices such as test driven development, rigorous time tracking, code refactoring, will be encouraged.
Sprint 5 - Retrospection

Sprint retrospection

Sprint #5

Abstract
This document describes the positive and negative practices performed during past sprints and define an action plans to improve the process.

Version

<table>
<thead>
<tr>
<th>Version</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialization</td>
<td>E. Girot</td>
</tr>
</tbody>
</table>

Highlighted practices

<table>
<thead>
<tr>
<th>List of factor that worked really well during the sprint just completed.</th>
<th>List of factors that were problematic during the sprint just completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refactoring went well thank to TDD.</td>
<td>Sprint log was unrealistic (not enough slack time).</td>
</tr>
<tr>
<td>Implementation of new feature went well thank to the time spent on designing gross software architecture.</td>
<td>Some tasks were not broken down enough (&gt;= 5h).</td>
</tr>
<tr>
<td>The re-priorization of tasks was made based on tasks value and the necessity of keeping working and potentially shippable software.</td>
<td>Some tasks included in this sprint did not represent highest value from the customer’s perspective.</td>
</tr>
</tbody>
</table>

Actions plan

- Tasks shall be broken down more finely.
- Slack time shall be increased.
- Sprint estimation shall be performed more rigorously.
- Beneficial practices such as Test Driven Development and “Just in time design” are to be encouraged.
19  Sprint 6 - Retrospection

Etienne GIROT

Sprint retrospection

Sprint #6

Abstract
This document describes the positive and negative practices performed during past sprints and define an action plans to improve the process.

Version

<table>
<thead>
<tr>
<th>Version</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialization</td>
<td>E. Girot</td>
</tr>
</tbody>
</table>

Highlighted practices

<table>
<thead>
<tr>
<th>List of factor that worked really well during the sprint just completed.</th>
<th>List of factors that were problematic during the sprint just completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful integration of new feature validated its design.</td>
<td>Impediments spotted in previous retrospective have not been removed.</td>
</tr>
<tr>
<td>Working software presents a satisfying robustness.</td>
<td>Some of the tasks scheduled do not represent immediate need.</td>
</tr>
<tr>
<td></td>
<td>Same tasks have been postponed from the last two sprints =&gt; They do not represent high value.</td>
</tr>
</tbody>
</table>

Actions plan

- Tasks shall be broken down more finely.
- Slack time shall be increased.
- Sprint estimation shall be performed more rigorously.
- Regular refactoring and code review have to be maintained on the same pace.
20 Sprint 7 & 8 - Retrospection

Etienne GIROT

Sprints retrospection

Sprint #7,8

Abstract
This document describes the positive and negative practices performed during past sprints and define an action plans to improve the process.

Version

<table>
<thead>
<tr>
<th>Version</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialization</td>
<td>E. Girot</td>
</tr>
</tbody>
</table>

Highlighted practices

<table>
<thead>
<tr>
<th>List of factor that worked really well during the sprint just completed.</th>
<th>List of factors that were problematic during the sprint just completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The slack time allocated to the sprints, permitted to complete all scheduled tasks.</td>
<td>Worked on unscheduled tasks to complete sprints.</td>
</tr>
<tr>
<td>Re-use of script “templates” allowed to save a significant amount of time.</td>
<td>Retrospections were not rigorously performed.</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>Tasks concerning SQL script implementation were overestimated.</td>
</tr>
<tr>
<td>-</td>
<td>Task concerning CSS adjustment was underestimated.</td>
</tr>
</tbody>
</table>

Actions plan

- Sprint retrospection shall be performed every week.
- Estimation must be improved to avoid working on unscheduled tasks even though being ahead of the schedule is less harmful than being late.
- Estimate of tasks similar to others achieved in the past must take into account the time saved due to the habit.
- Secondary tasks (graphical adjustment, documentation...) must not be underestimated.
- Consider gathering script templates, project templates, code snippets and conception guidelines to save time when reproducing tasks.
Sprints retrospection

Sprint #9,10

Abstract
This document describes the positive and negative practices performed during past sprints and define an action plans to improve the process.

Version
<table>
<thead>
<tr>
<th>Version</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialization</td>
<td>E. Girot</td>
</tr>
</tbody>
</table>

Highlighted practices

<table>
<thead>
<tr>
<th>List of factor that worked really well during the sprint just completed.</th>
<th>List of factors that were problematic during the sprint just completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The slack time allocated to the sprints, permitted to complete all scheduled tasks.</td>
<td>Sprint 9 was quite unpredictable because of the back and forth with the customer after the release (graphical adjustments, last minute statistics generation).</td>
</tr>
<tr>
<td>Planning poker allowed finding a consensus concerning refactoring, SQL query implementations and CSS that turned out accurate.</td>
<td>Retrospections were not rigorously performed.</td>
</tr>
<tr>
<td>Tasks were broken down and estimate separately after use of planning poker.</td>
<td>-</td>
</tr>
</tbody>
</table>

Actions plan
• Sprint retrospection shall be performed every week.
• The limit of 30% slack time per sprint must be drastically increased on release sprints to handle last minute adjustment while preserving sprint monitoring.
• Planning poker brought interesting input so far, keep using it and observing its effect over the next sprint.
Sprint 11 - Retrospection

Etienne GIROT

Sprints retrospection

Sprint #11

Abstract
This document describes the positive and negative practices performed during past sprints and define an action plans to improve the process.

Version

<table>
<thead>
<tr>
<th>Version</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialization</td>
<td>E. Girot</td>
</tr>
</tbody>
</table>

Highlighted practices

<table>
<thead>
<tr>
<th>List of factor that worked really well during the sprint just completed.</th>
<th>List of factors that were problematic during the sprint just completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task estimates are the result of a consensus based on technical considerations. (Planning poker)</td>
<td>Tasks could have been broken down more finely.</td>
</tr>
<tr>
<td>All the tasks were completed and no unscheduled task was performed.</td>
<td>Chart generator class significantly needs refactoring.</td>
</tr>
<tr>
<td>CSS task was given enough time.</td>
<td>Implicit subtasks were forgotten.</td>
</tr>
</tbody>
</table>

Actions plan

- Planning poker will be used on a regular basis.
- Chart class will be refactored before summer code-freeze when customer project’s are less urging.
- Developers will be encouraged to take notes about tasks singularities when these don’t appear on JIRA.
# Sprint 12 & 13 - Retrospection

Etienne GiROT

## Sprints retrospection

### Sprint #12-13

#### Abstract
This document describes the positive and negative practices performed during past sprints and define an action plans to improve the process.

#### Version

<table>
<thead>
<tr>
<th>Version</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialization</td>
<td>E. Girot</td>
</tr>
</tbody>
</table>

#### Highlighted practices

<table>
<thead>
<tr>
<th>List of factor that worked really well during the sprint just completed.</th>
<th>List of factors that were problematic during the sprint just completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good use of quick design sessions, proof of concept, prototypes and refactoring allowed for a clean and efficient development of new features.</td>
<td>Code comments are too few.</td>
</tr>
<tr>
<td>Tasks were broken down quite finely.</td>
<td>The CSS was included in the sprint even though we knew it wouldn’t be completed.</td>
</tr>
<tr>
<td>Estimates were accurate thank to planning poker and slack-time minimum limit.</td>
<td></td>
</tr>
</tbody>
</table>

#### Actions plan

- Code will be commented and short code guidelines and example will be made available to the team to ease feature understanding and usage.
- The tasks included in sprints whose sole purpose is to complete them need to be broken down into several subtasks if they obviously are going to exceed team capacity.
- Estimation process will be carried on.
24 Sprint 14 - Retrospection

Etienne GIROT

Sprints retrospection

Sprint #14

Abstract
This document describes the positive and negative practices performed during past sprints and define an action plans to improve the process.

Version

<table>
<thead>
<tr>
<th>Version</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialization</td>
<td>E. Girot</td>
</tr>
</tbody>
</table>

Highlighted practices

<table>
<thead>
<tr>
<th>List of factor that worked really well during the sprint just completed.</th>
<th>List of factors that were problematic during the sprint just completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The tasks that concerned the implementation of new projects based on templates and generic class re-use were precisely estimated and demonstrated a very satisfying time-to-market.</td>
<td>This sprints dealt with general improvement of the platform and internal needs and took place shortly before code-freeze, it has been rather an &quot;all you can code&quot; sprint than a sharply defined one.</td>
</tr>
<tr>
<td>-</td>
<td>Tasks were not precisely estimated and unscheduled tasks were performed.</td>
</tr>
</tbody>
</table>

Actions plan

- Good practices concerning estimation have been worth it and constituted the most effective lever for process improvement ensuring at the same time sustainable pace and productivity.
- Some kind of tasks tend to be recurrently underestimated, solutions need to be found to correct it.
- Good practices concerning development (refactoring, unit test, quick design sessions...) allowed our team to be highly reactive to customer needs, produce easy-to-maintain software as well as focus our efforts on valuable tasks only.
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Etienne Girot