Successfully bootstrapping a large scalable Scrum practice at Royal Dutch Shell

David Segonds
Technical and Competitive IT
Shell Global Solutions (US), Inc.
Houston, USA
d.segonds@shell.com

We will present the saga of a successful transformation from a struggling software development group to a scalable Scrum practice within Royal Dutch Shell. This group of sixty individuals encountered many obstacles on their journey to carry on the development of a large, 25 year old, legacy application. You will see how, over two years, we implemented a set of organizational, technological, procedural, and cultural changes to lead this group forward. Finally, we will present our vision to further strengthen and accelerate this value delivery system.

Enterprise Scrum, legacy software, organizational changes, quality assurance team bootstrapping, Scrum scalability, agile

I. INTRODUCTION

This paper describes the two-year journey, from January 2010 to December 2011, of the Subsurface Software Interpretation and Visualization group (SIVI) at Royal Dutch Shell (Shell). SIVI is composed of approximately sixty individuals, primarily located on two sites in Houston, Texas. SIVI is also contracting some of its software development operations to a couple of firms in Europe and in the USA. SIVI is part of the Subsurface Software directorate (SSW) at Shell.

A. The product

SIVI is delivering a complex set of functionalities that we will call GeoSigns/nDI[1] herein. Geophysicists and Geologists (G&Gs) in Shell are using those applications to interpret geophysical data and build models of the earth subsurface to better understand hydrocarbons reservoirs.

GeoSigns/nDI is built upon 123DI, the previous generation of proprietary subsurface interpretation tools. Both generations of tools share the same code base but while 123DI is restricted to operate on the Linux operating system, GeoSigns/nDI can also operate on Microsoft Windows operating system. GeoSigns/nDI has additional features that are the results of proprietary research conducted in other parts of Shell.

SIVI and its predecessors developed 123DI underlying technologies over the course of two decades. By 1997, 123DI was the *de facto* standard within Shell in North America. Between 1999 and 2006, the user base quickly grew to *innovators* and *early adopters* [2] all around the world. In turn, those individuals, in their respective regions, successfully championed 123DI to an *early* and even *late majority* of end-users as a replacement to third-party applications. At that time, SIVI was nimble and very responsive to end-user needs.

B. The challenge

However, during the first decade of this century, the needs of the enterprise changed. There was a strategic move to Microsoft Windows and a need to consolidate and productize some of the research conducted in other parts of the organization. In response, Shell started funding the development of GeoSigns/nDI in late 2007.

During the same period, it appears that the user population evolved and became more sensitive to flaws and quirkiness in internally developed software. Upper management expectations also changed and there was a need for more predictability in delivery and deployment.

Those new expectations were not necessary aligned with SIVI's *modus operandi* and SIVI embarked on a transformation journey from a small product development team to a professional software delivery group.

C. The ground truth

By those measures, in December 2009, after two years of development, the GeoSigns/nDI project was not doing well. Only version 1.0 had been delivered and it was so incomplete that only about thirty end-users worldwide were willing to play and experiment with it, compared to the thousand end-users for the previous generation of product. Work on version 2.0 was underway, but no delivery date had been set nor was any in sight. The team had a mountain of work to do. This included a daunting list of features that needed to be delivered under the programme funding in effect, but it also included a dismaying backlog of defects. To make matters worse, neither software features nor defects had been fully prioritized. Not only was the work somewhat disorganized, but the organization of the team itself needed improvement. The whole process used for software development, even though it was adequate in the past, seemed only to pretend to mimic software engineering best practice necessary to meet the new expectations. Programme governance was dysfunctional, seeming to provide only hindrance and no help to the project.

[1] *GeoSigns* is a registered trademark of *Shell Trademark Management BV*

As part of a company-wide reorganization, named *transition 2009*, every level of management affecting the project was replaced. The team feared for its existence. They knew that they were far behind on delivery aspirations, but they had been taxed by expensive decisions in which they had not participated, like the transition to a new geological database and move to the Windows Vista platform. The team felt that upper management had only criticized them for the adverse consequences of these and similar decisions.

Moreover, in early 2010, the Subsurface Software Engineering Excellent Group (SSEE) contracted Construx Software, a Seattle based company to conduct an organizational assessment of SSW. SSEE is another component of SSW whose purpose is to continuously improve software development practices for this directorate. This was the second annual assessment and the report highlighted that:

- Developer unit testing was widely used but often considered expendable under schedule pressure. System and integration testing was overly focused on narrow use cases and professional testing was largely missing.
- SSW's agile software projects were using a wide variety of practices and did not consistently include some recommended best practices for this type of project.
- Product backlogs were not always ready for development and excluded critical work items. The criterion for completion was not always clearly defined. Scrum teams were not always empowered as teams and teams were not making full use of sprint retrospectives.
- Effective project management relies on effective estimation practices. SSW's estimates did not account for all necessary work, product backlogs were not effectively estimated over the duration of a project, and estimates did not have sufficient input from the developers performing the work.

While this assessment was referring to Scrum [6], it is important to mention that SIVI, for the most part, was only very loosely following this methodology. For example, the daily Scrums were happening semi-weekly at best, iteration planning was often absent, and sometimes requirements were ready in the middle of iteration after the developers started coding. If SIVI were to implement a Scrum practice, it would have to rebuild from the ground up, and start by explaining to upper management that even though the team claimed to practice Scrum, they actually were not.

On a positive note, the assessment also highlighted that a number of the staff had worked on the same or related applications for numerous years and had developed an extensive understanding of how Shell's systems work. SSW had staff that joined from other parts of the company and brought with them important background knowledge about Shell's operations and user community needs. The level of support from G&Gs, and the depth and breadth of developer domain knowledge remained strong assets to meet business objectives.

At that point, it was clear that SIVI's mandate was not only to deliver a robust version 2.0 but also to develop long-term production capabilities to better serve enterprise and G&Gs community needs, while leveraging the extensive support from those G&Gs and its software developers' domain knowledge.

II. THE TRANSFORMATION

A. *Spring 2010: Early stages and version 2.0*

In February 2010, during a departmental workshop, the leadership team asked SIVI to commit to a release backlog for version 2.0, effectively narrowing the scope. Also, a quality criterion based on number of defects was agreed upon. During the first few months of 2010, SIVI focused on implementing this restricted scope and fixing defects. After a stressful few months, the team delivered version 2.0 in May 2010.

During that time, SIVI's leadership team, which as a consequence of *transition 2009* was mostly new, got acquainted with the group and attempted to formulate a plan of action.

The leadership team is composed of: a general manager in charge of SIVI; a product manager in charge of the product content and overall ownership through a team of G&Gs; a planning manager in charge of scheduling, finance, and communications to management; a deployment manager in charge of third-tier support, training, and worldwide deployment; and a software development manager in charge of software development and testing activities.

At that point, if little else, SIVI's leadership team decided to deliver versions on a strict schedule and because the resources were fixed, the scope for the project would have to be variable. This clarification of the *project management triangle* [1] may not seem like an important step but it helped set the scene for what followed and anchored many of the later initiatives. It also helped address the predictability that was expected by upper management.

B. *From May 2010 to November 2010: Version 3.0*

1) *Definition of done*

With GeoSigns/nDI version 2.0 delivered, the work mandated under the investment proposal from late 2007 was not complete and the conscious goal of the leadership team was to deliver another incomplete version by the end of 2010. This would help set the expectation that SIVI is able to deliver incremental versions to its end-users, albeit incomplete and not as robust as the majority of end-users would want for the time being. The leadership team was getting acquainted with the full scope of the investment proposal and it was not clear how much of the initial planned scope had actually been done or what *done* actually meant. Quite often, developers were first implementing features based on vague requirements and then went back—sometimes many months later—to fix defects.

In order to increase predictability and to transition towards an iterative development model, with a proper backlog, the leadership team deemed essential to formalize a *definition of*

done that would include acceptance testing, regression testing, and debugging.

2) ScrumMaster contributions

Early in this release cycle, the leadership team directed the development teams to produce burndown and earned value charts to assess the release cycle progress and increase transparency. This was a total failure as this reporting activity was impeding the teams' progress, only managing to increase their stress without providing usable results. It also clearly demonstrated the need for better planning tools beside the overly customized JIRA, a product from Atlassian, and miscellaneous spreadsheets that were in use at the time, to plan iterations and manage the backlog.

In July 2010, with the need to better understand the work of those teams without burdening them further with additional requests for reporting, the leadership team decided to contract with one ScrumMaster. The ScrumMaster was working with each development team, gathering data and providing necessary information to the leadership team on what the next steps in the implementation of an iterative software development method might be without getting in the way of those development teams.

Later that year, the ScrumMaster also trained SIVI to write stories in preparation of version 4.0. This exercise led to a better understanding of the backlog but was not as successful as expected due to the lack of a proper planning tool. An experiment on using index cards miserably failed because this approach is not scalable and cards cannot be accessed remotely.

While some elements of Scrum were now in place, namely the daily meetings, the definition of done, monthly iterations, and a ScrumMaster, many basic elements were still missing. Most notably, the size of the teams was inconsistent, and implementing a given feature, per the *definition of done*, required the intervention of multiple coordinated teams.

3) Running into trouble

During that time, the leadership team got a better appreciation for the backlog, figured out which teams were struggling to implement the scope, and which teams were not. The ScrumMaster and the leadership team quickly figured out that three development teams were struggling.

The first team (Team A) had trouble passing acceptance tests. This team was in open conflict with the G&G who provided the requirements. Without coordination with other teams, this team had decided on an implementation route that was involving a very high level of abstraction unprecedented in the application—i.e. unsustainable by the most senior developers on other teams. The ScrumMaster helped the team narrow the scope, decompose the work in smaller increments, and agree, with the impacted G&G, on acceptance criteria for each of those increments. In addition, the unorthodox implementation was rapidly abandoned.

The second team (Team B) was implementing an overreaching low-level change in the application. The team had started implementing this change a few years earlier, but they had abandoned this effort leaving part of the change in our Software Configuration Management (SCM) system mainline for a couple of years. However, this effort was necessary to handle onshore hydrocarbon reservoirs and because of this high value, the leadership team decided to plow forward and add additional resources to this effort.

The third team (Team C) was implementing an overreaching change in a separate branch of the SCM system—Subversion was used at the time. The leadership team decided to postpone this effort until version 4.0.

4) A burgeoning User Interface Automated testing effort

In parallel with those software development efforts, one individual, with the help of a couple of consultants, started experimenting with User Interface (UI) Automated Tests based on Squish, a testing framework published by froglogic. Using Squish, testers can record a workflow and specify validation points for this workflow. Then, either manually or as part of continuous integration, one can replay the recorded workflow. Squish launches the application, generates UI events, and for each validation point, compares the recorded state with the current replay. This tests if the application behaves as it did when the workflow was recorded. This was a promising effort that, once integrated in the continuous integration builds, helped us detect some regression defects within hours of the source code change that introduced them.

SIVI chose Squish for its UI Automated Tests needs because GeoSigns/nDI uses Qt, an open source project, as its UI framework and Squish is well suited to work with Qt applications.

5) The importance of Software Engineering Excellence

SIVI was not alone in its quest to implement a Scrum software development practice and received quite a bit of assistance from the Software Engineering Excellence group (SSEE).

a) Training

Among other symptoms, the three teams mentioned above had trouble estimating their backlog, and the requirement gathering process left much to be desired. These two problems were two concrete examples of an extensive need for software engineering basic training.

As part of a larger initiative, SSEE invited SIVI to participate in estimation training during which most of us learned about the notion of *cone of uncertainty* and the black art of software estimation [3].

SIVI also participated in several other basic training courses on such topics as requirements gathering, software inspections, or testing. Portions of this training extended to Subsurface Software (SSW) management so they could get familiar, if this was not already the case, with software engineering essential practices.

This training program helped establish a common understanding of our software development practice across all of SSW and help publicize our specific needs quite different from research or the traditional G&G practice at Shell.

b) Standards

Furthermore, SSEE helped us by publishing a set of standards. Those standards are part of our foundation for a successful software development practice and establish a minimum baseline for: configuration management, estimation, inspections, lifecycle, testing, security, risk management, requirements, design, construction, export classification, and third party development.

c) Health checks

Finally, SSEE is conducting yearly health checks by interviewing individual groups and assessing their performance around a variety of metrics. No group is ever always in the green and it helps identify weaknesses, develop mitigation plans and increase the overall effectiveness over time. Their assessments are getting stricter over time to incentivize continuous improvement.

6) The struggle to complete version 3.0

In October 2010, all developments teams except Team A and B, who were unable to complete the scope initially defined for version 3.0, stopped adding new features and started to fix known defects. Team A and Team B continued to work on their specified scope.

Our objective for version 3.0 was to fix all defects which resulted from that development cycle or defects with a high severity that had been introduced during version 1.0 or 2.0 development cycles. Unfortunately, this proved to be a colossal effort and by the end of November, it was evident that SIVI had a version 3.0 but it certainly was not meeting the quality criteria hoped for. Since, GeoSigns/nDI contained many of the features requested by our end-users, the leadership team decided to declare victory and deploy version 3.0, with the understanding that SIVI would continue to address more defects and stop implementing new features—except for team A and team B—until reaching the quality objective.

C. December 2010 to April 2011: Version 3.1

So, as SIVI concluded the year 2011, it was a mixed bag of results, and many challenges still remained ahead.

On the positive side, as the end of the year arrived, SIVI had a version 3.0 containing new features requested by end-users; clarity on the project management triangle, a definition of done; one ScrumMaster; some training on requirement gathering, story writing, estimation, and code inspection; an embryonic UI Automated Testing system running within our continuous integration system; daily Scrum meetings; less broken builds.

On the negative side, many individuals still had trouble decomposing epics into small-enough stories, and grossly underestimated development efforts. In addition, SIVI had a large backlog of defects, mostly non-cross-functional teams, and only one ScrumMaster.

The leadership team decided to first tackle the defect backlog.

1) Enhancement to the defect backlog management

a) Cleaning the backlog

In late 2010, SIVI had about 1,400 items in the defect backlog and a lot of them were not exactly actionable. Mixed with the true defects, one could find requests for enhancements, new features, or general comments. Once those non-defects were identified and removed, remaining defects were referring to non-existing datasets more often than not, or were missing clear steps to reproduce. SIVI engaged in a systematic clean-up, and worked hard fixing defects for three months and ended up with a clean database containing about 300 actionable defects.

Fixing or closing those 1,100 defects was only part of the solution. Without additional precautions, SIVI was risking falling in the same trap as before. It was imperative to maintain a clean database in the long run. To do so, G&Gs decided that a *defect monitor* would inspect all new incoming defects and mercilessly reject those that were not actionable. The defect monitor was to rotate every sprint.

b) Linear defect categorization, simple quality criteria

Each defects had four variables: *impact*, *usage*, *priority*, and *blocker*: impact quantified the effect of the defect on the application and could be set to high, medium, or low; usage related to the number of end-users affected by the defect and could also be set to high, medium, or low; priority was a combination of usage and impact so [high impact, high usage] defects had the highest priority while [low impact, low usage] defects had the lowest priority; finally defects resulting from regressions were flagged as blocker which could be set to *true* or *false*.

Our defect tracking tool, JIRA, was unable to compute priority given a set of impact and usage ratings. So one had to set priority manually whenever impact or usage was changed and this was not happening in practice. It was not clear how to handle regression defects (i.e. blockers) because their priority could vary greatly. In addition, impact and usage notions were only vaguely defined and subject to many debates, or even arguments. Finally, generating reports to assess the state of the defect database was arduous and difficult to automate.

After much debate, the leadership team decided to collapse those four categories—impact, usage, priority, blocker—into one called *severity*, whose possible values are clearly defined. The leadership team also settled on quality criteria, shown in table I. As indicated in that table, SIVI addresses defects in decreasing order of severity, but is leaving handling details to the teams, the G&Gs, and the ScrumMasters.

TABLE I. QUALITY CRITERIA

Severity	Quality Criteria	When to fix
Regression	Zero	As soon as possible; preferably before the end of the current sprint
Critical	Zero	As early as next sprint
High	Zero	As early as next sprint
Medium	Up to 50 defects	As early as next sprint
Low	No limit	As early as less than 50 medium defects are present

This approach helped us better manage the defect backlog, simplify communication, and help signal to all that SIVI needs to meet the quality criteria at the end of a release cycle, but also at the end of iterations.

The five possible values for severity are:

- Regression: This is a problem preventing a successful build (such as broken automated tests) or a defect for a function that worked before the current iteration. In summary, a regression is something that used to work and is now broken;
- Critical: Defects that cause disastrous consequences for the system in question such as critical loss of data, critical loss of system availability, critical loss of security;
- High: Defects that cause very serious consequences such as severely broken or incorrect functions or algorithms, or broken functions that interrupt an important work flow and that have no identified workaround;
- Medium: Defects that cause significant consequences; A defect that needs to be fixed but there is a workaround, such as a badly broken function but with a known workaround;
- Low: Defects that cause small consequences but that are easy to work around, or trivial defects that cause no negative consequences and produce no erroneous outputs. Examples include misleading error messages, displaying output in a font or format other than what the customer is expecting, simple typos in documentation, bad layout, or misspellings on screen.

By March 2011, SIVI successfully met the quality criteria and delivered version 3.1 which was more stable than any other GeoSigns/nDI or 123DI versions before. In addition, team A and B completed the scope initially planned for 3.0 and delivered it with 3.1.

D. April 2011 to December 2011: Version 4.0

It was nice to see version 3.1 out of the door but it was now time to address some of the more systematic issues and fine tune the delivery machine that SIVI was becoming.

1) Team Reorganization

a) Development Teams

Up to that point, the teams were predominantly organized horizontally. SIVI had a database team, an infrastructure team, a business logic team, a user interface team, a 3D graphic team, etc. It also had some teams organized by feature but they were depending on the horizontal teams to complete part of their stories. This created some dependencies and hindered story estimation as the work was split among separate groups. Finally, some teams had ten individuals while other teams had only one or two persons.

After careful consideration, the leadership team reorganized the development group to promote cross-functional teams. Now, in most instances, each team is able to implement stories from beginning to end and therefore minimize inter-team dependencies.

b) Quality Assurance team

Not only did the leadership team want to reorganize the development team, they also wanted to increase the role of Quality Assurance in the delivery process.

At that point, it is important to note that regression testing and acceptance testing was solely the responsibility of the G&Gs internal team—reporting to the product manager—while debugging was the responsibility of the development teams. As exhibited during the Construx Software assessment in 2010, there were no professional testers or testing team *per se*.

Scrum advocates embedding testers in each development team. While this model works fine when professional testers are already present in the organization, the leadership team did not find it suitable when the first objective was to acquire the necessary skills to appropriately test the application.

The leadership team selected the individual who experimented with UI automated testing in 2010—see II.B.4) above—to be the QA team leader. The QA team developed its own charter and hired a mix of SME, UI automation testing specialists and professional testers to kick start the testing effort.

Within a few months, the amount of testing greatly increased. The UI automated tests went from just a handful to over one hundred. Automated tests ran each night for a few hours—the equivalent of approximately forty man-hours of manual testing. This helped detect defects much earlier than ever possible and helped populate the regression defects backlog.

The QA Team also conducted some manual regression testing in consultation with developers to target areas that could have been affected by version 4.0 development activities.

This team also coordinated the User Acceptance Testing (UAT) at the end of the release cycle to accompany and survey end-users who tried out release candidates.

Overall, these efforts saved a very large amount of G&Gs time, freeing them to focus on writing requirements and accepting newly implemented stories.

So, testers are not embedded in each development team but they provide shared services that benefit the whole group.

c) The role of team leaders

Scrum treats teams as whole entities and does not make distinction among the individuals in a given team. However, the leadership team decided for practical reasons, that one individual in each team would be the team leader. Indeed, with forty individuals, it is not practical, for developers, testers, and continuous integration engineers to all report directly to one person.

All team members participate equally in the three Scrum ceremonies but team leaders participate in additional meetings such as risk management or dependency meetings that aim at increasing communication and facilitating the overall development process.

For assigning stories and defect backlog management among teams, SIVI primarily operates within a pull model where team leaders decide which team will handle which stories. Sometimes, team leaders have difficulties deciding which team should handle a given backlog item. In that case, the product manager and the software development manager make the assignment decision.

Team leaders also provide *exploratory* and *budget estimates* [3] when grooming the backlog or planning for a release cycle. The team members provide the *commitment* estimates when they decompose stories into tasks during the planning meeting at the beginning of a sprint.

Since SIVI is outsourcing some of its development work to third parties, Team leaders are also responsible for defining and approving work proposals; and accepting source code deliveries.

d) ScrumMasters

So, after this reorganization, SIVI ended up with five development teams and one QA team. The leadership team decided to hire an additional ScrumMaster and let the two ScrumMasters decide, in agreement with the team leaders, who would be facilitating the work of which team.

The ScrumMasters play an essential role in safeguarding the Scrum process. The ScrumMasters help the team estimate their work, manage their day to day workflow, and identify dependencies and risks.

But, above all, ScrumMasters monitor the backlog and provide relevant metrics for teams to operate and leadership team to make informed decisions.

e) Product Owners

As previously stated, SIVI's G&Gs are reporting to the product manager. They are the subject matter experts for GeoSigns/nDI and they provide requirements for new features to development teams.

In practice, development teams have the skills and knowledge to implement feature requirements coming from multiple G&Gs, and a G&G can give their requirements to multiple teams. In other words, there is no one-to-one relationship between G&Gs and development teams.

In effect, this situation led to a competition among the G&Gs to lobby for the importance of their stories and to request bandwidth from a development team to implement those stories.

In case of a tie, the product manager had final say on the priority of requirements. However, in June 2011, one development team was dealing with six G&Gs and because the team wanted to satisfy all those individuals in parallel. They ended up working on many different stories concurrently rather than sequentially. This was an extreme case, and probably an anti-pattern. It clearly demonstrated that SIVI's approach, consisting in the product manager breaking ties between G&Gs, was not sustainable.

Nevertheless, SIVI still needed to have the option to distribute requirements from a G&G to more than one team and teams to handle requirements from more than one G&G. SIVI needed to alleviate the product manager workload and therefore attempt to distribute the responsibility of prioritizing requirements among its development teams.

SIVI ended up adopting the following approach: A given G&G can have two roles: In the role of a *product owner*, the G&G is responsible for managing the development team backlog, grooming it as necessary, and prioritizing incoming stories; in the role of a *feature owner*, a G&G is responsible for providing requirements and writing stories.

Exactly one G&G is taking the role of product owner for a given team and a G&G can be a feature owner for multiple development teams.

This also implies that a product owner for one team can act as a feature owner for his development team or for other development teams.

This arrangement, clarified the role of product owner, and helped us spread the product manager prioritization-responsibilities on multiple teams.

Conflict over product owner and feature owners for a given team may still occur but they are rarer and product owners act as Cerberus for their team. This smoothes out the flow of incoming stories and generate healthier prioritization debate. The product owner still has to intervene on occasion, but development teams are more self-sufficient as a consequence of this change.

2) Backlog management and metrics

Up to that point, SIVI was still using a mix of spreadsheet, heavily customized defect tracking system, and word processor documents to keep track of the backlog. This was highly inadequate and during the second quarter of 2011, SIVI started using Team Foundation Server (TFS), a Microsoft product, to track the feature backlog. While the need for a better planning tool had been identified as early as April 2010 by the leadership team, SIVI had to wait for the enterprise TFS implementation to ramp up. SIVI augmented TFS planning capabilities by using Urban Turtle, a brand developed by Pyxis Technologies.

For many months, feature backlog was in TFS while defects remained in JIRA. This was far from an ideal situation but SIVI needed time to get familiar with TFS and to migrate the defects from JIRA to TFS.

Given this effective way to manage our backlog, the two ScrumMasters started gathering team velocity information, publish SIVI sprint burndown chart. Admittedly, it took a little bit of time for the development teams to accept that burndown charts and velocities were a way for them to better estimate and optimize their sprint commitments, and not a way for management to gauge their productivity.

3) Fixed four-week sprints

SIVI was using monthly sprints but the leadership team considered them to be impractical as it was tricky to schedule Scrum ceremonies from sprint to sprint. For example, should one schedule the review meeting on the last day of the month or the first business day after that? In addition, most recurring

routine meetings occur weekly or on a semi-weekly basis in Shell. For those reasons, the leadership team needed to adjust the sprint length.

It was important to adopt the same sprint length for all development teams as they are working on the same product and this requires a high level of coordination [5].

Since Scrum [6] recommends a sprint length shorter than 30 days, the leadership team decided on four-week sprints.

Four-week sprints are close enough to monthly sprints; they have the same benefits of a monthly sprint without the scheduling headaches. Shorter two-week sprints were also considered but it would have been another big adjustment for the team.

Since this decision, some team leaders have suggested shorter iterations while others would like to have the flexibility to modify the iteration length. For scalability and coordination reasons, the leadership team decided to stick to four-week sprints for the time being.

Development teams are celebrating the end of each sprint with a trip to a local restaurant or a group outing where they mingle with other teams.

4) The Rules of Scrum

In the fields of education and operations research, the Dreyfus model of skill acquisition is a model of how students acquire skills through formal instruction and practicing. Stuart and Hubert Dreyfus proposed the model in 1980 in an influential, 18-page report on their research at the University of California, Berkeley [7][8]. The original model proposes that a student passes through five distinct stages: novice, advanced beginner, competent, proficient, and expert.

In 2010, while coaching Scrum teams at Landmark Graphics, an Oil & Gas independent software vendor, Jaron Lambert and Simon Orrell, applied the Dreyfus model to Scrum [4]. In April 2011, they gave a seminar on this topic and freely shared the rules they had been using. Slightly modifying this initial set, leadership team and ScrumMasters developed the SIVI rules of Scrum.

At first, this generated some heated debate and discussions in SIVI and some team members saw the rules as a scourge. However, those rules clearly established a common base for all the individuals in the group by clarifying Scrum basics and easing internal communications. As the group is climbing the five-stage ladder of skill acquisition, Scrum rules are less mentioned, but they are still in effect, and they played a critical role in cementing SIVI Scrum practice.

5) A few mishaps before a resounding success

That release cycle was not without a few hiccups which are worth mentioning even though they are not as severe as the problems encountered in 2010.

a) Too many changes in a short period of time

Every year, Shell conducts a global survey to gauge employee morale and satisfaction. The latest survey, conducted in May 2011, shows that initiating many changes over a short period of time during spring 2011 created quite a bit of stress for the staff and many complained of a lack of autonomy during that period.

b) The difficulty to maintain a low defect count

SIVI started this release cycle with about 300 defects in the database and this number slowly crept up over the months to reach about 450 in September. While some in the leadership team advocated for a continuous debugging effort, there was also pressure to complete the backlog. As a result, the release cycle had to end with a couple of sprints dedicated to debugging.

c) Haunted by ghosts from the past

As you may remember, Team B struggled to implement an overreaching low-level change in the application during the version 3.0 release cycle and they completed the scope with version 3.1 in March 2011.

However, during the summer 2011, the leadership team received some alarming reports from two asset teams who were experiencing some difficulties with the new features implemented by Team B. An asset team is a group of end-users who are working on a specific area of the world in search of hydrocarbons.

Those two groups were under pressure to complete their work using GeoSigns/nDI and SIVI had to send developers and G&G on site to analyze and fix the remaining issues. This negatively impacted the version 4.0 scope.

d) Success and celebration

Finally, thanks to the team reorganization exercise, better backlog management, velocity measurements, regular sprints, and rules of Scrum, and a lot of hard work, version 4.0 development cycle ended up being a great success in December 2011. This latest version not only contains the initial scope funded in late 2007 but also some additional industrialized research that was not mature enough when the initial scope was defined. SIVI invited some retired staff to join one hundred other guests to the celebration.

III. LESSONS LEARNED

I am part of the leadership team whose peregrinations are described in this article. Based on this journey, and if faced with similar challenges to those encountered by SIVI in 2009, I would keep the following lessons in mind.

This experience taught me that it is critical to clarify the *project management triangle* when approaching a software development effort. This is fundamental but easily forgotten when facing external and internal pressures. Often, end-users care a lot about predictability which means that the schedule has to be fixed. Since the resources are often also fixed, in order to bring an effort to completion on time, splitting the scope in multiple manageable development cycles is necessary.

We also discovered that this was not enough because even if the scope is flexible, it is important to know when a given feature is complete or not. A *clear definition of done*, early in the GeoSigns/nDI development effort would have benefited end-users and SIVI development group.

I made the mistake of wanting metrics too early in the process. Burdening an already stressed group of individuals with providing measures for the process was not a wise decision. However, contracting an agile coach or, in our case a ScrumMaster to gather data and get the pulse of the development group was useful. This helped us diagnose problematic situations more precisely.

I suffered from the lack of a proper planning tool well until mid-2011 and with a large group of people working on one project with numerous dependencies, this is an essential change. Index cards can help some but this is not a scalable solution. I would try to bring a planning tool earlier in the transformation process.

Each development team has its own personality, often linked to its leader's personality. For example, we found a different set of circumstances—internal and external—leading to the struggles of teams A, B, and C. Each team had to be addressed separately since their challenges were specific. However, we found necessary to set a common cadence for multiple teams, such as the four-week sprint duration or the rules of scrum. One of the most successful teams has considered this cadence to be an imposing bridle. Managing a large group of people working on the same application is a balancing act between the need for common processes and the need for autonomy.

We found that providing a common set of rules, getting our inspiration from the Dreyfus model of skill acquisitions, has helped established a commonality among the teams and the multiple trainings also helped in that regard. Each team has to find its autonomy within the group akin to an individual in a scrum team.

The sprint duration change, the team reorganization, and the rules of scrum deployment really had a negative impact on morale. Not only did the teams have the mandate to deliver but they had to do it within a set of rules for which they did not provide input. At this point, I am not sure if making all those changes at once, close to the beginning of a release cycle was a mistake or not. On one hand, this had a negative impact on morale but on the other hand, one year later, it seems that most individuals have not only adjusted but are thriving.

Over the course of a few months, and as a result of retrospectives and dedicated discussions, a self-selected group reviewed and enhanced some of the processes that had been initially imposed by the leadership team.

I still believe that a *top down approach* is necessary in some cases because of urgency but those decisions must be accompanied by clear *communication* and *commitment* to approve later changes when they benefit the whole group.

I have also learned that forming a team dedicated to QA and testing is effective. My natural instinct was to follow the scrum approach and embed a tester in each development team but a dedicated team can be effective in quickly raising the skill level for the whole group.

In our case, setting a *quality criterion*, based on the number of defects has been effective in slowly increasing the quality of the overall product. However, it requires clear and concise severity definition, and an appropriate defect triage process emphasizing defect actionability. We are not yet convinced of our system effectiveness.

Lastly, I must admit that I was confused on the best way to handle the product owner role. Our product owner, and feature owner roles have been a struggle to establish initially but as a result, our teams operates more effectively.

Now that we have a scalable scrum practice, we are putting it to the test. We have engaged in two separate efforts that have not born fruits yet: we created a dedicated User Experience team which will help us acquire more skill and knowledge in the domain that we are unfamiliar with; we are working on understanding how to effectively let our end-users preview our software at the end of every sprint. Both those initiatives leverage the lessons learned during this two-year journey and involve more collaboration than a bare top-down decision from the leadership team.

ACKNOWLEDGMENT

I want to thank Tom Riddle and Daniel Fremion for their unflagging support; Alan Jackson, Doug Tudor, and Rene Villafuerte for battling with me on many occasions and setting me straight in quite a few; Kevin Wright, Andrew Kennedy, our ScrumMasters, for their unabated efforts; SIVI, SSEE, and Bluware staff, for their hard work, ingenuity and dedication to our success; Our contracting partners for their timely and high-quality work; Construx staff for enlightening me in many occasions and providing invaluable advice along the way.

Without all those individuals, there would be no journey, and no success.

REFERENCES

[1] M. Newell, G. Marina, *The Project Management Question and Answer Book.* New York, NY: AMACOM, American Management Association, 2004, p. 8.

[2] G. Moore, *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers.* New York, NY. HarperBusiness, 1999.

[3] S. McConnell, *Software Estimation: Demystifying the Black Art.* Redmond, WA: Microsoft, 2006.

[4] S. Orrell, J. Lambert, *Applying the Dreyfus Learning Model to Focus your Coaching Approach. http://program2011.agilealliance.org/event/4eb7866112a4b6b96f44e328e9885a3a, August 2011*

[5] E. Woodward, S. Surdek, M. Ganis, *A practical guide to Distributed Scrum.* Boston, MA: IBM Press, 2010, pp. 58–59

[6] K. Schwaber, J. Sutherland, *Scrum Guide*. http://www.scrum.org/scrumguides/, October 2011

[7] S. Dreyfus, H. Dreyfus, *A Five-Stage Model of the Mental Activities Involved in Directed Skill Acquisition*, Washington, DC: Storming Media, February 1980.

[8] S. Dreyfus, "The Five-Stage Model of Adult Skill Acquisition," Bulletin of Science, Technology and Society, vol. 24, issue 3, June 2004, pp. 177-81