

# CSM Practicing Certification Renewal Assessment

**Name: Anupam (Alex) Singh    email: alex\_singh\_us@yahoo.com    date: 2005-03-22**

Scrum depends on the inspect and adapt mechanisms of process control to manage the complexity of projects. For inspection to work, everyone must know what is being made visible. To implement the Scrum process, such regulating mechanisms as defined roles, involvement versus commitment, time-boxes, and regular cycles are used.

## **1. Describe one project on which you have used Scrum over the last twelve months.**

### **Purpose**

The project was intended to deliver accounting data online, to 500+ clients, via:

- Pre-generated monthly and annual PDF accounting reports,
- Parameterized reports that could be invoked on-demand.

One of the major goals of the project was to see major improvements in operational efficiency by reducing the involvement of the Client Services group with answering common client questions and with report generation and delivery.

### **Length**

The original project was budgeted for 9 months. The team spent three months pursuing a delivery strategy that had to be abandoned due to the high cost and long delivery schedules involved. I got involved in the project when the team had already spent a little more than 2 months investigating the strategy -- it was immediately apparent that we needed to come up with an alternative solution if we were to meet the delivery deadline. We then spent 1 week going over our options, ironing-out possible kinks with the favored approach, and convincing the business and management that a change in tack was required.

### **Cost**

The project had an initial budget of \$1.2 million (approx. 72-person month duration) that was increased to \$1.7 million when management realized that the original funding was too low.

### **Value**

The benefits of the project were manifold:

- The company had just lost a big client that generated over \$1 million in profit for our business unit. In addition, a few more clients were at risk due to the inadequacy of the company's offering as far as reporting on the investment performance was concerned. Improving client satisfaction, by better addressing their trust accounting reporting needs, was a necessity.
- The company could, in addition, save money by helping the clients find the information online that they would call the client service representatives for. The goal was to see a 75% reduction in client calls within six-months of the release.
- If successful, the implementation could additionally be used for delivery of other types of reports

### **Size**

The project team included 6 programmers (Java, IBI, and JSP), 3 DBAs (1 for design and 2 for loading data and creating the views), 1 full-time business analyst, 1 full-time analyst/tester, one part time Documentum and one

part time IBI consultant. This was in addition to a ScrumMaster and a Product Owner. We also contracted with a User Experience consultant who worked on the project for one sprint to nail-down the UI. We had 3 financial analysts who clarified the requirements and helped specify the test cases.

## **Teams**

The team was broken down into smaller sub-teams: DBA, infrastructure, and MVC. The DBA team was responsible for migrating data from the existing Oracle DWH to a new UDB data mart with instances in Dallas, TX and Merrimack, NH. The infrastructure team was responsible for ensuring that (1) our new report generation functionality (IBI WebFocus) and supporting infrastructure worked end-to-end, and (2) the new document storage application (Documentum) was flexible enough for our needs. The MVC team was responsible for creating and publishing the reports.

The sub-teams were cross-functional to a large degree. The DBA team was a separate dedicated team as we realized that getting the basic schema right the first time was critical; it is not easy to keep modifying the schema once data has been loaded in. Once the schema was in place, the team worked closely with the MVC team on implementing the reports.

The infrastructure team worked closely with consultants from Fujitsu (familiar with Documentum) and IBI. The team designed and implemented the Documentum storage and also implemented the reporting infrastructure.

The MVC team was responsible for all aspects of the MVC framework -- Java and Struts (design and implementation of classes and services), IBI reports, JSPs, etc. Once we had validated the basic architecture (in the initial zero feature release), the team members self-organized and chose the functionality they would work on. Apart from designing and implementing the reports, team members were also responsible for working closely with the DBA group to (1) ensure that required data, for reports being worked on initially, was populated first, (2) tweak the schema, views, and tables as required to get the reports to work and perform adequately. The MVC team also leveraged the services created by the infrastructure team for document storage and retrieval.

## **Initiation**

Selling the notion of iterative development to management was not too difficult. When I took over, we had already lost 3-months and had just 6-months to deliver. Management liked the idea of gaining visibility into what was going on and not having to wait till the end of the project to realize that dates would not be met. They also agreed with getting the testers and the users involved sooner in the process, so that their feedback could be incorporated in the next sprint -- this would save considerable time in the formal "Systems Acceptance Testing" and "User Acceptance Testing" phases. We also decided to pay lip service to the formal Summit Ascendant requirements for a: System Design Specification document, an Architecture Implementation and Planning document, and a Technical System Design document -- we would create a light-weight combined document that simply described the basic architecture discovered during the initial no-feature Sprint.

In the kick-off meeting with the whole team, I told everyone present that we were going to use an iterative process for delivering the functionality to the users in chunks. I gave them an overview of Scrum and also indicated that we would be adhering to some of the development practices encouraged by XP (simplicity of design, continuous testing, daily builds, etc.). I decided to stay away from pair programming as management would not buy-in the concept of two people working on a single machine; I did move people around so that people who would naturally be working closely together had adjoining cubicles.

In a follow-up Sprint planning meeting we went through the prioritized product backlog items and selected a

sprint backlog from that. Backlog items were written in the form of user stories on index cards (these were later put into an Excel based Scrum workbook).

## **Change**

Team morale was pretty low as developers had been put through the wringer on their previous project -- they had to work late daily and on weekends to put in code that they were not comfortable with and then to go through fire-drills fixing the bugs. The reasons were that requirements would be changed at the last minute and management still expected the changes to be implemented in the original time-frame.

To address the need to be more flexible and to produce better quality code, we decided to use Scrum and XP. XP led the team to investigate tools such as Cruise Control, Cactus, and Grinder; the team also recognized the benefit of writing jUnit test cases as it gave them the confidence to make changes in their code as necessary. Keeping things simple was a revelation to the team; they had been writing bloated code as the users had always wanted everything.

The introduction of the concept of business value and how it related back to each of the requirements (cost/benefit) forced the business users to immediately prioritize the backlog items into critical, high, medium, and low categories. The development team would then work with the Product Owner on further clarifying the critical and high priority items.

The developers felt the full impacts of the changes too. Agile development forced the team members to recognize that they not only had the responsibility but also the accountability for whatever they were working on. They could no longer blame some other team -- everyone was in the thing together. The team also had to get used to turning things arounds quicker; gone was the luxury of taking things easy while the various requirement and design documents were being written and reviewed. The most impressive change was in how quickly the team improved in the estimates they provided -- this could have been due to the cross-functional nature of the team as the requirements were better understood and the team could decide where to implement some functionality (be it the data base or the report) to get the task done faster and in a cleaner way.

## **Reporting**

To meet the different demands of the corporate finance department and the business unit's management team, I decided to go with a hybrid process:

- We were mandated to use the traditional waterfall method (Summit Ascendant) as that was the corporate standard by which all projects were evaluated -- a sudden change to a completely agile development approach would not have been politically feasible for our business unit.
- But to meet the delivery deadline (we were already 3 months behind schedule) we needed a more flexible light-weight approach.
- We ultimately decided that we would develop the functionality in monthly increments that the business could then provide feedback on. In addition, we would also create the documents required by the Summit process, but would considerably pare them down.

As the Scrummaster, I would provide two flavors of the dash-board reports. For the corporate group, I would run the numbers off the company wide time Mercury tracking system and present it in the required format; for the business unit, I would present the backlog, the sprint goals, the sprint backlog, and the sprint burndown chart at the end of each sprint. In addition, the team would release working software to the Web site at the end of each sprint.

## **Management**

The Product Owner was a VP in the Finance department. She knew the accounting systems inside-out and she was quite successful in fulfilling her role. She took the lead in writing the user stories, in prioritizing the various backlog items, in specifying the tests that should be run for a given set of reports, etc. She was also instrumental in extracting information (logic) out of the third-party vendor that provides the accounting system we use. She was ably supported by a couple of talented analysts with significant fund accounting experience.

## **Engineering**

When we started the project, we did not have proper development procedures in place. There was no concept of continuous integration and frequent unit testing. We started requiring jUnit tests for all Java code – we didn't go as far as encouraging test-driven development, though. We also encouraged developers to check in all their code at the end of the day and to make sure that the day's final build compiled and passed the basic smoke-test. Towards the latter end of the project, we started evaluating the CruiseControl framework for a continuous build process and Cactus for unit testing server-side java code (Servlets, EJBs, etc) with the intent of lowering the cost of writing tests for server-side code.

Even though pair programming was not a goal, we encouraged developers to communicate and work together more often to discover better ways of implementing a given functionality. Clean and simple code was required and coders were encouraged to refactor their code. Frequent code reviews were conducted to help improve developers' code. People soon saw the benefit of writing unit tests and they became more confident of making major code changes when required. Writing unit tests is now a requirement throughout the business unit.

## **Stabilization**

No stabilization sprints were required. Making sure that the functionality worked in the system integration environment was a part of the Sprint; the production environment mirrored the lower level environment and all we had to do was to make sure to promote all changed content to production on Thursday night and perform a smoke-test. We would hold the Sprint Review meeting on the last Friday of the month where we would demonstrate the application to all interested parties.

## **Scrum Process**

People worked in typical cubicles spread out throughout the same floor. Eventually, we moved people around so that everyone would be together -- not in an open space, but in adjoining cubicles. We also appropriated a small meeting room that we used exclusively for putting up charts, backlogs, etc., and for holding the daily Scrums.

The standard Scrum process was modified a bit on this project:

1. All product backlog items were written as user stories and were estimated using the 50% and 90% confidence levels as described by Mike Cohn.
2. We did not have 100% cross-functional teams. The infrastructure team and DBA team started out as separate teams; over time the DBA team started working more closely with the MVC team.
3. We also had weekly meetings with the Product Owner and with the two analysts who worked for her. This gave us an opportunity to get feedback quicker (rather than having to wait until the end of the Sprint).
4. Our "sprint goals" were very generic: "implement 5 reports this Sprint".
5. For the most part we worked on the higher priority items first. The only exception was the assets and liabilities summary report (the most important report) that we had to work on last – we couldn't do this until we had actually worked out the underlying detailed reports first.

6. After each Sprint Review meeting, we conducted a quick meeting to talk about what worked, what didn't work, and what could be changed.

## **Success**

This project was very successful. The success of the project and the methodology employed are apparent from the fact that additional funding has been provided for subsequent releases. The backlog has additional functionality that will take an estimated 8300 hours to complete; also, feature delivery has been scheduled for a release every month. The architecture implemented was extended to support the delivery of other report types (not just accounting reports) to clients and was also used to serve as the basis for a new Intranet site for internal users.

## **2. How do you cause the accuracy of Product Backlog estimates to improve? To what degree does their accuracy matter?**

At the start of the first Sprint, we were provided with very inaccurate estimates by the developers. Apparently, the developers had never been trained on estimating tasks – they usually gave wildly optimistic time frames and understated the effort required by a factor of 2 or more. This made estimating what could be developed in a Sprint pretty hard. I noticed that they usually did not account for all the things that could go wrong and that they also did not really understand what the requirement actually meant from a coding perspective.

As an exercise, I had the team work through a couple of requirements. We used stickies to jot down all the tasks that would be needed to deliver the requirement as specified. We also identified dependencies and potential risks that could affect the development. I noticed that the estimates improved dramatically, when the team estimated together. Over the next few sprints, the estimates got better as the team members became more adept at identifying additional tasks that needed to be done to fulfill the original requirement.

During our sprint retrospective we then examined any significant discrepancies. We then laid out our original assumptions and tried to determine where we got tripped up – major impediments that were not accounted for, things taking longer than expected due to dependencies that were not apparent, etc.

The team learned from each Sprint, and used the gained knowledge to better estimate tasks. Backlog items would be compared to previously executed work of a similar nature and estimates would then be adjusted accordingly.

## **3. How do you cause the accuracy of what a team commits to for a Sprint to what the team actually delivers?**

The Sprint burn down chart was the main tool used for tracking estimated work to what actually got produced. The product backlog by its very nature is a rough estimate; we classified items as Tiny, Small, Medium, Large, and Gigantic. These corresponded roughly to 1, 2, 4, 8, and 16 story points.

The product backlog item estimates were then refined during the sprint planning meetings and adjusted up or down based on new information/insight. We always attempted to use the previous Sprint's velocity to (1) determine what we could take on during the next sprint, and (2) re-estimate the release duration.

## **4. What metrics do you use to track the development process? Which metrics have been changed, removed, or newly implemented as a result of using Scrum?**

The only metrics being used during development was an estimate of how much was done – what's the percent completion. This was a completely useless metric as you could be 90% done with nothing working exactly how

the users wanted. In addition, the percent complete figures were derived from inaccurate figures that the developers came up with. During latter stages of the project, the metrics were testing related – number of new bugs introduced versus number of bugs closed.

The use of the burn down chart and the use of Expected Business value made the existing metrics moot.

**5. What type of training, resources, or tools would best help you successfully employ Scrum in the future?**

Integration with Microsoft Project to produce the types of reports that management used to the traditional project management methodology expects would be helpful.

**6. (Optional) Scrum and Extreme Programming are sometimes used together. What must be considered when this is done?**

Each of the 12 core practices of XP must be considered for their suitability to an organization. Some companies are very traditional and will balk at things like pair programming. In addition, XP leads one down the path of self-discovery; this could be a painful process for some. Pair programming or even working closely with other team members can make visible the skill level of an individual. This is sometimes of concern as the individual in question may not have the expertise that he/she claimed she had.

In my opinion, it's always easier to sell what people want to buy. Throwing a lot of new terminology at people may scare them off, but rephrasing in more common terms may do the trick. Using the term XP may cause management to roll their eyes, but identifying the XP practices and explaining how it really is common sense and how it will prove beneficial is helpful. So instead of talking about Continuous Integration, talk about how it makes sense for people to continually validate their code changes to ensure that they didn't break something else. Instead of YAGNI, discuss why it is not such a good idea to develop and then maintain code that may never be used. People usually get the point.