Lessons Learned in Academic Hybrid Software Development Projects – A Retrospective

Robert F. Roggio*, Dalila Castilla
School of Computing
University of North Florida
Jacksonville, FL USA 32224
*Email: broggio {at} unf.edu

Abstract— **According to some researchers, a hybrid approach can help to optimize the software development lifecycle by combining two or more methodologies. The Rational Unified Process (RUP) and Scrum are two methodologies that successfully complement each other to improve the software development process. However, the literature has shown only few case studies on exactly how organizations are successfully applying this hybrid methodology and the benefits and issues found during the process. To help fill this literature gap and to provide a major development project for a five-person team of first year software engineering graduate students at The University of North Florida, the Lobbyist Registration and Tracking System for the City of Jacksonville (COJ), FL was designed and implemented using a hybrid approach that integrated RUP and Scrum within IBM's Collaborative Lifecycle Management (CLM) solution.**

While it is safe to generalize and assert that it is typical for software development projects to encounter challenges both in the corporate community as well as in academe, the hybrid development approach using RUP, Scrum, and CLM in an academic setting presented some unique issues. The purpose of this paper is thus to convey the distinctive issues arising from the hybrid approach in an academic setting to provide empirical evidence of these problems with suggestions as to how they might be avoided. The details of the year-long development project are reported in thesis form.

Keywords-RUP; Scrum; agile development; academe'; iterations

I. THE PROBLEMS

A. Team Composition and Work Schedules

The two-semester graduate-level project was undertaken at The University of North Florida (UNF) as part of the software engineering track within the Master of Science in Computer and Information Sciences degree program. The class had approximately 18 first-semester students, who were divided into three teams of six students. Team 1 developed the lobbyist tracking system. (No special significance was assigned to team numbers and no one had a problem with being called Team 2 or Team 3.)

Team 1 consisted of five individuals (one dropped within the first week of class). Of the five, two were well-experienced developers who worked in UNF ITS (Information Technology Services), which handles academic and administrative computing university-wide. Although new to the graduate program, their skills were quite good and they had software development experience within the framework of the systems that the university develops and maintains. The other three individuals were similarly new to a graduate level program, but had no experience in practical software development other than coursework. They also had no experience in the core technologies selected to develop the system.

Because of the rigidity of semester scheduling and the need to develop software quickly and on time, there was little time for these students to learn classroom best practices of software engineering while concurrently learning development environments, complex tools, experience very diverse team dynamics, and merge all these characteristics into a cogent successful development team environment. Regardless, they took their roles and supported other projects' activities required in order to deliver an application that met the client's expectations and requirements.

As it turned out, the team worked quite well together and did not experience problems that oftentimes development teams encounter. In retrospect, this was likely attributable to the maturity of the individuals to pool their differing levels of expertise. In particular, the two experienced individuals took on the bulk of the design and programming efforts, while the other team members provided database design, interface design, undertook documentation of use cases, developed and ran test scripts, ensured traceability and similar activities - skills that could be obtained essentially from course work.

Given this high degree of cooperation, serious scheduling complications nevertheless arose – not unique to team projects. But these arose from several levels. The class met twice a week in the evenings from 6-7:15pm. The experienced team members worked from 9-5pm and were thus unavailable during the day for development team meetings; the other team members were full time graduate students and were available during the days, but all had families and

spouses, which made meeting during the evenings other than class nights very difficult. Because the class met two times a week, this only left hours after class, which effectively meant the team usually remained after class after a long day. Management of the team, that is, the delegation of responsibilities was equitably determined, and each team member played a primary role, such as project manager, primary programmer, quality control individual, tester, and database developer. Each also had a secondary or backup role. The management and integration of these activities during available times (and all team members could not always meet at available times), made the coordination of team efforts very difficult.

Coordinating team member times for meetings is not an unusual problem in development teams in an academic environment. But another difficulty arose in trying to meet with the clients, all of whom were City of Jacksonville employees, who worked daily until 5pm. Students could not meet as a group during the day in downtown with the clients and the clients could not easily drive on to campus during the day. All modern development techniques agree on the critical importance of essential customer – developer relationships.

During the next academic year, the two-course software engineering course sequence was scheduled to meet once per week from 6 - 8:45pm. It was envisioned that this schedule would leave more available times for team members to organize their meeting times.

B. The Collaborative Life Cycle Management (CLM) Solution

The CLM is an extremely powerful tool. It covers all elements of the software lifecycle, and the capabilities for keeping everything together while providing a comprehensive solution is really outstanding. But it is not easy to learn especially for students who lack software development experience. Further, learning this solution and effectively using this solution with its vast array of features while concurrently learning principles and practices of software engineering is daunting even for the most ambitious student. While tutorials are available, the lack of prior experience with the CLM together with the students learning basic software development principles via the Rational Unified Process (RUP) and digesting the management of the development process with Scrum [2] proved to be a formidable challenge.

Tech support within the School of Computing at UNF often wrestled with providing a stable environment for CLM. CLM was hosted on a UNF computer, which proved to be reasonably effective. Yet many of the features inherent in the CLM, such as burn-down charts, velocity charts, and using the vast facilities within CLM to manage and track development and testing had to be eschewed by the development teams in order to meet the time constraints of iterations and sprints. Dates for these deliverables together with examinations, presentations, and more, often precluded exploiting some of the real power of the CLM.

C. Client Project Manager (PM)

Initially the software engineering professor for this course sequence met with the lawyer in the Ethics and Compliance Office to learn about requirements. Representatives from this office later met with students one time for a serious question / answer period. From this meeting and from additional meetings between the professor and the Ethics and Compliance Office, use cases were developed – but by the development team (not by the client). Development started once a reasonably solid set of use cases were developed and approved by the client for comments. It was not until well into the second semester that the COJ established a Project Manager to oversee what the students were doing from a client-perspective. Unfortunately, 60'% or more of elapsed project time had been spent. More effective client representation was sorely needed. This was not the way to proceed.

D. Changing Requirements

Perhaps the once constant in software development is change. All modern development environments embrace change and certainly the RUP [1] and managing development via Scrum support this reality. While experienced developers know that change will occur and expect it and react to it, it is always bit unsettling especially for new developers, who want requirements to be firm, complete, and unchanging. But they learn very quickly. Requirements that are “must haves” suddenly become changed and generally more complicated and comprehensive. This is frustrating to a student who is looking at the end of a semester dates, exam schedules, and closure. But for sure, change occurs.

As it turned out, most of the “must haves” in the lobbyist application remained solid, but the team did experience some “requirements creep.” A number of 'nice to haves' crept in too. These may often seem rather simple, and indeed sometimes they are. Oftentimes they are not. Despite the experience on the team, the first stage of the project (about one semester) consisted of five iterations of about two weeks each once the project got started; the second stage, which was partially based on the RUP (elaboration) but essentially development (used Construction and Transition for recognizing artifacts / work items) was planned to be five sprints based on Scrum. Three more sprints were added to implement the ‘nice to haves’ and schedule deployment activities (see Figure 1).

E. Version Control and Politics

Unfortunately**,** politics and other unforeseen circumstances sometimes occur. This development effort was no exception. When the project was nearing completion and demonstrations had been provided to the client who applauded the fine work and seemingly everyone was happy, versioning and politics come to center stage.

While universities technologies often seem to lag as compared to the industrial sectors, the converse seems to be true when operating or working with state or city

IBM Rational CLM Solution													
Stage I – RUP							Stage II – Scrum						
Inception				Elaboration			Construction				Transition		
Iteration #0	Iteration #1	Iteration #2	Iteration #3	Iteration #4	Iteration #5	Sprint #1	Sprint #2	Sprint #3	Sprint #4	Sprint #5	Sprint #6	Sprint #7	Sprint #8

Figure 1. Hybrid Approach Conceptualization

government. There appears to be a major lag in technologies often due to the economics of versioning. Such was the case in this development effort. The university was using ASP.NET MVC 4.5 and SQL Server 2008 and had been demonstrating the project for several months with the quiet assurance that handover would be uneventful. UNF had developed the software and database with versions to which the COJ had not yet upgraded adopted.

II. PROPOSED SOLUTIONS

A. *Team Composition and Work Schedules*

1) What we did: Class was scheduled two nights a week: 6 – 7:15pm. Some team members worked 9-5; Clients worked 9-5; other team members were available only during the days.

2) Better ways to go: Schedule the class once a week thus freeing up more available times within which the team can meet.

Get a firm commitment from the client for pre-established meetings: some on campus; some off campus at client shop. But these must be agreed to ahead of time and these need to be tagged, "Can't Miss Meetings."

Must have a client available for telephone calls, emails, Skype, or other forms of communication within the day when questions arise. Questions need answers in near real-time and cannot often wait for convenience. This requires a major commitment from client and developers.

B. *Collaborative Lifecycle Management Solution.*

1) What we did: In that the two developers on the team were familiar with Team Foundation Server (TFS), TFS was used. Products were then imported into the CLM. This was done in the interest of time and the need to eschew learning CLM. This is not the way to proceed.....

The use of CLM [3] must be a total buy-in at the beginning of the project and at least one or two must be the master of this solution. S/he must be able to provide guidance and learn how to use the vast array of capabilities within CLM and advise the team as to these capabilities.

2) Better ways to go: The professor instructor is much more familiar with CLM for this coming year than he had been in the past. Too, a student not in this course developed a thesis wherein he looked very carefully into Rational Requirements Composer (RRC) and how to use the use-case ability within CLM to capture use cases and to trace activities from that point forward. Team members must use the facilities within RRC.

The professor must insist on using CLM as the framework for the solution. He/she must not allow TFS or any other lifecycle management solution to become integrated with CLM.

Technical support must preload a version of CLM onto local servers and obtain a necessary number of licenses so that this hurdle need not be addressed once the semester (or quarter) begins. This is essential. The environment must be stable, and both the instructor and technical support must agree that this is the case at the beginning of the semester.

C. *Client Project Manager (PM)*

1) What we did: Because there was no designated project manager from the COJ supplied, the development team essentially developed our own iterations and sprints and back-fit the development into the two-semester project. We met with the client a time or two, but the client was an end user and had very little knowledge of the impact such a system would have on the COJ environment once ultimate handover took place. She was delightful but did not possess software development expertise**.**

2) Better way to go: While this might appear to be obvious, it is not always easy to have a single point of contact from the systems point of view to be available. But this is definitely needed. The PM took the bull by the horns, so to speak, and forced the development team to designate deliverables and ultimately handover. Further, she also took on a dominant role with the Product Backlog oftentimes realigning priorities for the development team, which turned out to be excellent and the way things should be.

The customer representative must be able to act for the client-side and prioritize features cited in the Product Backlog in order for the development to proceed in a customer-oriented prioritized manner. As it turned out, the PM was very aggressive. And, while the team had a number of conference calls, she always provided minutes of the calls and emails of 'understanding' which proved to be essential. She would capture the content of the meetings and assert these via email. This proved to be critically important to the team in that we could no always physically meet with the PM. The conference calls coupled with follow up confirmation emails was a good move. More meetings and conference calls are great bearers of fruit. The more face-to-face meetings that can occur, the more power the communications and the assurance that the development team is meeting client expectations and that things are proceeding on track. This point of contact is a must, and the authors strongly recommend that this point of contact as well as a definite schedule of F2F meetings and scheduled conference calls be established up front.

D. *Changing Requirements*

1) What we did: Perhaps due to inexperience and perhaps due to the length of semesters, and perhaps due to expectations, the development of iterations and sprints for the two stages reasonably encapsulated the two semester span time. This seemed reasonable. But it did not allow for change. Because the PM did not become actively engaged from the COJ, the team was plodding along feeling reasonably secure despite the heavy development responsibilities. Once the project manager became a major player (and again, this is essential), things changed and formal meetings / conference calls and **c**hanges were suddenly identified. The team did not allow for change when planning the development effort. Taking the product backlog and the sprint backlogs (assuming little changes) allowed the team to plan the workload out such that the workload mapped into the semester.

2) Better way to go: Plan on two additional sprints. If they are not needed, great. But they likely will be needed.

The software engineering program at UNF requires a software practicum course that is used by various faculty members for various reasons. It is a catch-all course. The team elected to extend the project into the short six-week software engineering practicum course during the summer. This provided flexibility to accommodate three additional sprints that the team used. This worked out very well and is an alternative to merely planning for two additional sprints - if the time allows.

E. *Version Control and Politics*

1) What we did: So, UNF developed a good project that was incompatible with the software on which it was to run at the City level.

Undaunted, the development team downgraded the development to be compatible with an older version of the software only to later learn that this too was insufficient to accommodate handover, despite all the discussion, preparation of user manuals, and more.

2) Better Way to Go: While this might appear to have been avoidable, the politics and poor communications sometimes exhibited during development did not foreshadow this development. So, clearly, compatibility of development versions and implementing versions of software between the developers and the clients must be pre-established. Needless to say, to discover after over two semesters of work that the product would not be implemented did not set well with the developers.

III. RETROSPECTIVE

A. *Team Composition and Work Schedules*

Plans must be made for solid, reliable communications between team members and their individual schedules and those of the client. Dates / conference calls and all mechanisms of communications must be pre-determined as well as what the focus of these communications is to be. Agenda must be planned; feedback accommodated. Dates set and met.

B. *Collaborative Lifecycle Management (CLM) Solution*

There is so much power in these approaches, and IBM's CLM is outstanding. However, that said, it is an imperative that some expertise be developed by one or more team members so that the real collaborative power and great capabilities within CLM can be brought to bear. Perhaps a local professional can come to your class and present some of the basic methods on how to use Rational Requirements Composer, Rational Team Concert, and Rational Quality Manager. There are a number of good tutorials, but our recommendation is to have a local person with expertise in CLM present some of the basic techniques on accessing the components of CLM. While an application system may certainly be developed without CLM, application lifecycle management (ALM) tools that encompass a comprehensive number of tools, charts, methodologies covering all phases of lifecycle development as well as management and tracking of these processes legislates the use of an ALM. Our recommendation is the use of IBM's CLM. Our personal experience can attest to the power of this solution [4].

C. *Client Project Manager (PM)*

Be certain to have a single point of contact as a client. The PM needs to be well-established. The authority / responsibilities that this person possesses must be articulated as well as the role s/he may play. The PM is the representative of the client, and all matters of requirements, prioritizing features for development as well as validation, change requests, handover, and more need to be centralized in this individual.

D. Changing Requirements

These are simply a fact of life in software development. The development team needs to be made aware of this fact, but more importantly scheduling is best served to have a little flexibility in deliverables.

E. Version Control and Politics

Versioning and control must be assured early on in the project. If there is to be a problem, then any risks associated with this potential must be mitigated early - very early in the development effort. Also, ensuring that all the team members have some knowledge or acceptable level of experience using the selected technologies assures a more productive environment and avoids schedule delays on the project related to this issue.

IV. CONCLUSIONS

Developing applications is not for the faint of heart. But a well thought out development scenario with appropriate tools (both development and management) plus clear relationships with clients and their environment as well as the constraints on the developers themselves is crucially important.

REFERENCES

[1] P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2003.

[2] R.T. Nishijima and J. G. Dos Santos, "The Challenge of Implementing Scrum Methodology in a Traditional Development Environment," International Journal of Computing & Technology, vol. 5, issue 2, 2013, pp. 98-108.

[3] M. Göthe, C. Pampino, P. Monson, K. Nizimi, K. Patel, B. M. Smith, and N. Yuce, Collaborative Application Lifecycle Management with IBM Rational Products, IBM Redbook, IBM Corp, 2008.

[4] D. Castilla, "A Hybrid Approach Using RUP and Scrum as a Software Development Strategy," Masters' Thesis, University of North Florida, August 2014.