Integrating Agile Development Methodologies into the Project Capstone – A Case Study

Christopher G. Jones[1]
Business Computer Information Systems
Utah Valley State College
Orem, Utah 84058-5999 U.S.A.

Abstract

Regardless of program concentration (System Development or Information Security), all four-year degree Information Systems & Technology majors at Weber State University (WSU) in Ogden, Utah, are required to take the capstone project management course. Not only does this mean student preparation levels in system development are uneven due to program emphasis but with recent changes in the Weber curriculum, students have mixed backgrounds in development methodology spanning SA/SD and OOAD. This case study describes an attempt to use an agile methodology to bridge the differences in background by focusing on a common set of analysis, design, and implementation artifacts.

Keywords: agile development methodology, Crystal Clear, eXtreme programming, IS 2002.10, light-weight systems analysis and design, project management, senior project capstone

1. INTRODUCTION

Weber, a large state university located in the U.S. intermountain west, offers undergraduate degree programs in Information Systems as well as an MBA with an IT focus. Students select from one of two concentrations: System Development or Information Security. All students, regardless of concentration, take the Senior Project capstone course.

Over the past couple of years, the Information Systems & Technologies Department (IS&T) has gradually introduced object technology, primarily through coursework in object-oriented programming using Java. To date, however, the Systems Analysis and Design course (a prerequisite for the Senior Projects capstone) only includes coverage of traditional system development methodologies such as Structured Analysis/Structured Design (SA/SD). Plans for the 2003-2004 academic year include a move to a new text with a balanced presentation of traditional and object-oriented system development approaches. This paper presents a case study exploring an approach to teaching the Project Management Capstone course with students having multiple system development methodology and programming language backgrounds. The paper begins by exploring the challenge of a mixed methodology capstone, introduces an Agile Development methodology as a possible solution, and then chronicles deployment of Crystal Clear, a lightweight systems development framework. A series of student retrospective exercises are used to evaluate the effectiveness of the approach. The paper concludes with lessons learned from the use of agile methods for small-team information systems development.

[1]jonescg@uvsc.edu

2. THE CHALLENGE

Although students registering for *IS&T 4730 Senior Project* may have met the course prerequisites, background preparation levels vary by programming language and development methodology. Most students have taken coursework in visual programming (VB 6.0) and conventional design (SA/SD) using data modeling (entity-relationship diagrams (ERDs)). Students in the Information Security concentration take additional coursework in networking and data security while students in the System Development concentration are exposed to web development. Approximately 30 percent of the Senior Project students have taken classes in object-oriented programming with some exposure to the Unified Modeling Language (UML) for program design. None have had experience with the increasing popular light-weight development methodologies. While all students have been exposed to heavy-weight approaches such as SA/SD or Object-oriented Analysis and Design (OOAD) with UML, few have had any practical experience with either.

Course content for IS&T 4730 follows IS '97 (Davis et al. 1997) and IS 2002 (Gorgone et al. 2002) model curriculum coverage of Project Management and Practice. Using project management concepts and foundational knowledge in systems, network, and database design, students work in teams to develop a significant computer-based system for real clients. At the end of the semester, students demonstrate their completed (and working) systems to the client and members of the faculty.

The semester time constraint for the course presents a formidable challenge. How do you teach 11 learning units on IS 2002.10 project management (Gorgone et al. 2002), form project teams, help students find and build a relationship with a client, oversee the development of entirely new systems, and provide tutoring on two completely different system development approaches in only 15 weeks? One approach is to front-load the course with the Project Management content and let the students develop at will. This was the strategy taken Fall 2002.

By the fifth week, the class had covered (a) introduction to project management, (b) project integration management, (c) project scope management, (d) project time management, (e) using project management software, (f) a review of structured analysis structured design and object-oriented analysis and design, (g) project cost management, (h) project communications management, and (i) a comprehensive project management case. The remaining topics (project quality management, project human resource management, project risk management, and project procurement management) were sprinkled through the remaining 10 weeks.

Three problems arose. First, since the first five weeks of the course were devoted to project management instruction, when students turned their attention to development they found it difficult to complete a project from start to finish in only 10 weeks. Second, the quality of the analysis, design, and implementation artifacts generated by students to document the development process varied dramatically and were often inconsistent within the two methodologies (i.e., SA/SD; OOAD). Third, motivation suffered as students came to view project documentation as an "after-the-fact" exercise to satisfy course requirements rather than as an aid in the development process itself.

In summary, the challenges facing WSU as it sought to deliver a successful capstone experience in project management and practice were:

- Mixed development backgrounds with some students exposed to SA/SD and others to OOAD
- Uneven preparation levels as a result of differences in the program concentration selected
- Little practical experience with the heavy-weight textbook methodologies, whether traditional or object-oriented
- Project artifacts that placed a drag on the project momentum rather than providing additional impetus

3. AN AGILE APPROACH TO SENIOR PROJECTS

Utah is home to the Agile Development Conference sponsored by the Agile Alliance in cooperation with the ACM Special Interest Group on Software Engineering (ACM SIGSOFT) and AITO (Association Internationale pour les Technologies Objets). This annual conference is "aimed at exploring the human and social issues involved in software development and the consequences of the agile approach to developing software" (Agile n.d.). The Agile movement is best known for eXtreme Programming, a light-weight approach to software development popularized by Kent Beck (Beck 2000). One of the founding members of the Agile Alliance, Alistair Cockburn, resides in the Salt Lake City, Utah area and hosts a monthly Salt Lake Agile Group. It was at one of these monthly meetings, that the group was asked to help address the challenges of the WSU IS&T 4730 senior project class, specifically the disparate development backgrounds and the burdensome documentation requirements of heavy-weight methodologies. The group responded by suggesting Crystal Clear – an "ultralight-weight" software development methodology created by Cockburn while working at IBM (Cockburn 1997, 2001).

Crystal Clear

"Crystal" is a family of development methodologies tailored to project criticality. "Crystal Clear", (circled below in Figure 1 as D6), is designed for one-to-six person teams working on projects with low to medium criticality, where project failure would not hurt the organization's bottom line.

According to Cockburn (Cockburn n.d.), the essence of the Crystal Clear approach to software development is:

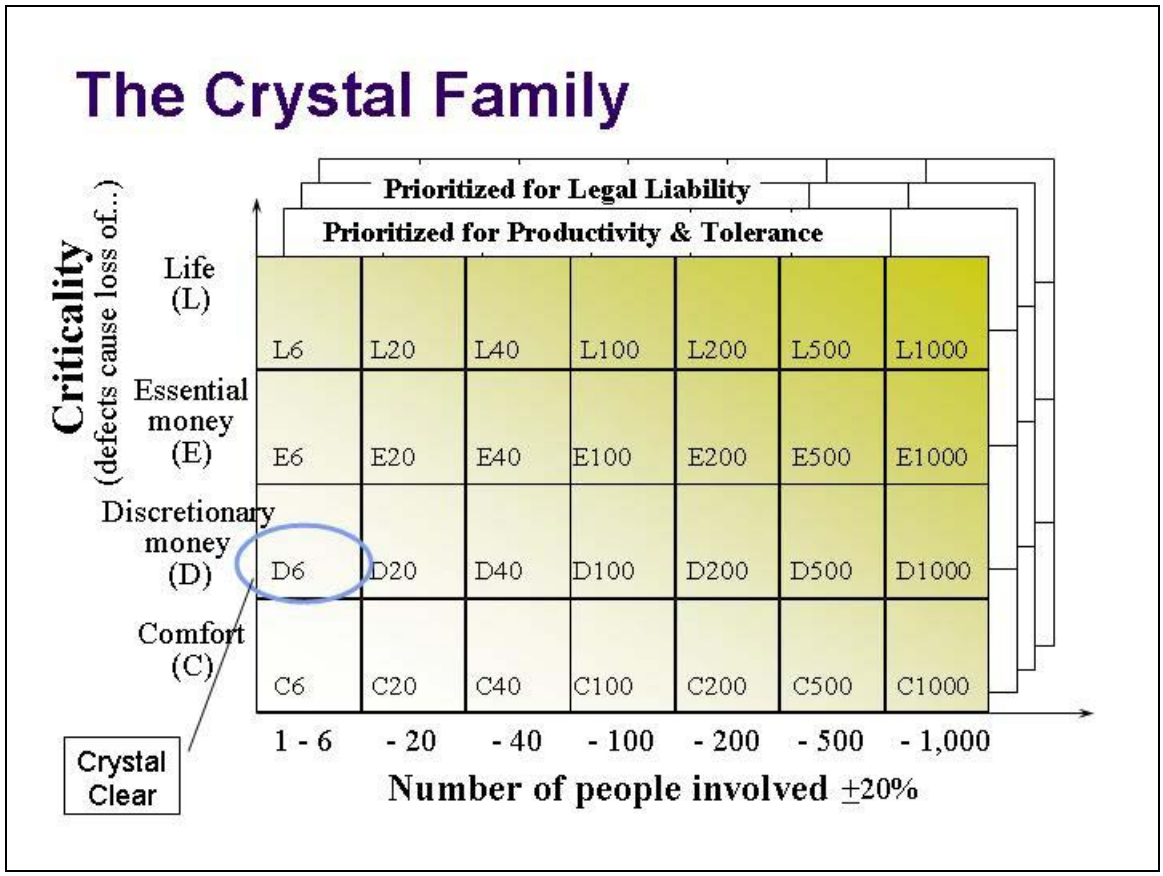

Figure 1 Crystal Family of Agile Methodologies

> The lead developer and two to five other developers in a large room or adjacent rooms, with whiteboards (preferably printing whiteboards), access to key users, distractions kept away, delivering running, tested, usable code to the users every month or two, periodically reflecting and adjusting their working conventions (p. 7).

Crystal Clear requires a minimum set of work products (WP) (Table 1). Of the 20 project artifacts, only WP 13 Common Object Model is currently paradigm specific. Cockburn (personal communication, January 10, 2003) readily admits WP 13 can be generalized to all domain data models including semantic data or entity-relationship models. For purposes of the Senior Project course, WP 13 was modified accordingly and relabeled "Common Domain Model". The only other modification to the work product list was WP 20 User Manual. Instead of focusing solely on a separate physical document, the definition of the WP was expanded to comprise all user assistance including online manuals and paperless help systems.

With the minor generalizations to the Crystal Clear methodology mentioned above, it appeared that the approach might well address the course challenges of mixed paradigms and the overhead of heavy-weight development methodologies. For this reason, Spring semester 2003, the modified Crystal Clear methodology was used as the standard development approach for two sections of the Senior Projects capstone. Section 01 included seven students organized into two project teams; Section 02 included 18 students in five project teams. Students were allowed to develop their Senior Project using either SA/SD or OOAD. All submitted work products were identical except WP 13 Common Domain Model. SA/SD project teams submitted entity-relationship diagrams while OOAD teams submitted Unified Modeling Language (UML) class diagrams.

Agile Development Exercise

To provide students with an end-to-end overview of lightweight development methodologies, an exercise in agile development adapted from Bergin's eXtreme Planning activity (Bergin 2000) was conducted early in the semester. Working in teams, students designed and "built" a hot drink maker to the specification of their "customers", played by a few members of the class. For purposes of the exercise, "built" meant that the development team was to draw a picture of the desired machine, incrementally. Students developed high-level Use Cases to capture system requirements, worked with their customers to identify functional priorities, and prototyped the system with pencil and paper drawings under short time constraints. Throughout the exercise, student teams could consult with the customer to clarify specifications and validate prototypes.

At the end of the first 25-minute iteration, students were asked to reflect on the process and modify team activity accordingly. At the end of the second iteration, students were asked to reflect on the agile development exercise as a whole, answering such questions as:

- Did the developers build what the customers said they wanted?
- Were the customers happy with the result?
- Were deadlines met?

Responses varied by section. In Section 01 of the Senior Project class, each developer team met or exceeded customer expectations, whereas in Section 02, only one team of the four made the customer "happy". The primary difference between those teams that satisfied the customer and those that did not was that teams satisfying the customer tended to include the customer as part of the development team, referring to the customer for clarification often.

Overall, the agile exercise was an appropriate learning activity for helping students understand the high-communication levels required of lightweight development approaches. Students really enjoyed the process, bringing incredible energy and enthusiasm to the simulation. The only change to be made to activity would be to reschedule it earlier in the course rather than during week four.

Table 1 Crystal Clear Work Products

No.	Work Product	Description
1	Mission statement	A brief description of the system to be built including its purpose and value in the larger context.
2	Team structure	How the team is partitioned to accomplish its work, and what the lines of agreement and reporting are.
3	Development methodology	The methodology consists of the roles on the project, the team structure, the process the team follows, the work products they maintain, how they review their work and how closely they do that, and the skills they need among themselves.
4	Release sequence	A statement or dependency diagram showing, briefly, what the order of the software releases is and what is in each.
5	Viewing & release schedule	For each increment, describes when the users will get to see, and get to use, the sections of a new system as it comes out in stages.
6	Risk list	A list of the top risks facing the project, their likelihood, and fallback plans.
7	Project status	A set of annotations on the project plan. It contains a selected set of needed accomplishments, which ones are completed, which ones are underway, and some measure of stability, certainty or progress toward each.
8	Actor-goal list	What types of people/organizations/computer systems/automated systems will drive the system, or directly care about seeing the function of the system enacted. What their goal is for each of their interactions, at several levels (strategic/summary goals and task-level goals). Correlation between primary actors of the system and goals the system supports.
9	Annotated use cases	The functional requirements for the system. A fully dressed Use Case is written with one of the full templates, identifying actors, scope, level, trigger condition, precondition, and all the rest of the template header information, plus project annotation information.
10	Requirements file	A collection of information indicating what it is that is to be built, who is intended to use it, how it provides value, and what major constraints affect the design. Includes Stakeholder List; In-Out Scope Table; Problem Domain Vocabulary; System Interfaces and Technology Constraints; Development Processes; Human Backup, Legal, Political, and Organizational Issues; and, Preliminary Project Plan.
11	System design	A large-scale, low precision description of the overall system, from a technical perspective.
12	Design sketches	Draft drawings of the system, its components, and their relationships. Notes regarding design trade-offs and decisions.
13	Common object model	For the problem domain, the set of entities or events and their related attributes and behaviors. The relationships between those entities/events.
14	Screen drafts	Screen prototypes, screen navigation flow, and report prototypes.
15	Source code	Commented program logic, organized according to the underlying development paradigm.
16	Packaged system	The application source, compiled code and/or interpreted code and the required configuration information, files, directory structure, or registry data. May be componentized into a binary library, bundled into a self-describing archive, or packaged into self-installing compressed format.
17	Migration code	All one-time code written to migrate existing application data and behavior to the new system. May involve data conversion and writing wrappers to encapsulate legacy functionality. Often written in scripting languages.
18	Test cases	A set of inputs and expected results that exercises a component with the purpose of causing failures.
19	Defect reports	Documentation of any event that occurs during the testing process which requires further investigation.
20	User manual	A document describing the application user interface, such that someone unfamiliar with the system can use it.

Reflection Workshops

One of the key differentiators between traditional and agile approaches is the emphasis on rapid adaptation of the methodology *during* development, in response to changes in the project (Abrahamsson, Salo, Ronkainen, and Warsta 2002; Thomas, n.d.). Cockburn refers to this aspect of agile development as "periodically reflecting and adjusting their working conventions" (Cockburn n.d., p. 7). As part of Crystal Clear, project teams are encouraged to pause periodically to conduct "Reflection Workshops." Key to these workshops is an on-going evaluation of team structure, team process, and working conventions. Team members are asked to identify what they would like to keep, where they are having problems, and what they would like to do differently in the next phase of the project.

Following each major project milestone, students in both sections of Senior Project conducted reflection workshops. Each student was asked to complete a one-page retrospective on the completed milestone. In addition, students were asked to comment on the Crystal Clear approach.

Analysis Retrospective: Most students felt the team roles they had adopted were working well and that the communication channels were effective. The biggest problems faced were scope creep, shared access to team work products, divergent views and personality differences, and the time required to learn the technology specified by the client. For the design phase, students felt they needed to increase the number of team meetings, clarify team member assignments, establish a team repository for work products, and increase communication frequency from weekly to daily.

For the Systems Analysis milestone, students were to submit WP 1 through 10. With respect to the Crystal Clear methodology, students found the structure of the work products helpful but somewhat burdensome. A few students reported that they didn't like the readability of the Crystal Clear text. Typical student comments included:

> The one thing that I would like to comment on is we have a lot of assignments. I'm not saying that they aren't useful, but they're time consuming and take a lot of the focus away from actually working on the project. I know they're necessary, but it is hard to complete all the work products while trying to meet as a group to complete the project. All of our schedules are so different and when we meet together, it's stressful because we're trying to get everything done.
>
> Have less deliverables and more time in class to work on the actual project. The work products allow us to visualize and I can see when this is more productive in a large project, but we only have 3 months. Feels a lot like busywork, yet I can see how this is applicable.

Design Retrospective: Students again reported that they felt the team roles (after some role reassignments) were working well and that the development process was effective. Increased communication levels and regular team meetings had improved information flow. Major problems included getting all team members to project meetings, occasional communication lapses, and a desire to expand the project to include more technology than the customer requested.

For the Systems Design milestone, students were to submit WP 11 through 14. As for Crystal Clear, most students felt the design documentation requirements weren't as heavy as those for the systems analysis phase. Some students felt the Crystal Clear text and the supporting PowerPoint slides provided by the instructor did not include enough examples of the various work products:

> Crystal Clear is OK, but I wish that there were more specific examples. Some of the deliverables aren't clear.

Implementation Retrospective: During the final phase of system development, students reported that team dynamics, especially team commitment, enabled the groups to complete the project on time. Several students commented on how their teams "came together nicely", putting in "long hours at the last minute to deliver a

working system." A few teams praised the agile methodology for its adaptability, enabling them to "change quite a few parts of the system" late into system construction. Communication, regular meetings, and a balanced team composed of members with complementary skill sets were frequently mentioned as important drivers for success during the system implementation phase.

Key problem areas during systems construction revolved around time, technology, and team meeting synchronization. Even with incremental deliverables with some part of the project due each week, students felt crunched for time near the end of the course. "It took longer than expected" was common a theme, usually followed by "we underestimated the amount of time needed." Students reported trouble with familiar technology ("we had a difficult time with DB connectivity") and issues with new technology ("unfamiliar language"; "platform variations"). Finally, as the teams began to meet more frequently to address deadline pressures, many team members reported "trouble synchronizing our schedules."

For the Systems Implementation milestone, students were to submit WP 15 through 20. As the section of the draft Crystal Clear textbook covering these artifacts was incomplete, a series of PowerPoint slides was developed to provide examples of each work product. For the most part, students had little trouble with commented source code. However, because they had minimal experience in actual deployment, several project teams struggled with migration code and systems packaging. Inexperience was also an issue in testing. Several project teams approached test plans and defect reports as an exercise rather than as an integral part of the development process.

Crystal Clear Retrospective: At the end of the course, students were asked to assess Crystal Clear as an agile methodology and provide suggestions for improvement. On the whole students appreciated the adaptiveness of the agile approach but felt the methodology could be improved by reducing the documentation requirements. Typical comments included:

> I feel that this methodology is great because it gives you direction and help and yet is flexible to let you change things along the way.
>
> I really like the Agile concept. At times, it seemed like there was too much documentation, but most of it was fairly necessary.
>
> I think near the end of the course is when we got most of our work done because we weren't concerned with finishing paperwork. I know the documents are important, but it was difficult to get things done. Class time really helped us to organize and take care of the administrative tasks we had to do.
>
> I really liked the Agile development methodology. I felt like it presented enough questions to truly analyze the project, without it being weighed down in paperwork. I hope to use it again.

4. LESSONS LEARNED

Integrating an agile approach into the Project Management course is not only possible but students appreciate the flexibility of a lightweight methodology. The ability to modify the development process midstream, even under tight time constraints, enabled project teams to deliver high priority functionality to the user by course end. All seven teams deployed working code that met or exceeded client expectations. Key lessons learned from this course experiment were:

- *Although agile methodologies evolved from the Smalltalk community, they need not be object-centric.* A few simple generalizations to Crystal Clear allowed students, regardless of background in SA/SD or OOAD, to generate a fairly common set of analysis, design, and implementation artifacts. In fact, except for WP 13 Common Domain Model, all documents were universal.

- *Even with a reduced documentation set, students still perceived the Crystal Clear work products as "busy work" rather than absolutely essential to the development process.* A possible explanation for this could be due to the

structure of course assignments. Although Crystal Clear only requires 20 artifacts total, students were expected to complete draft versions of many of the work products. For example, students were asked to submit low-precision annotated Use Cases (Use Case Briefs) one week and then the following week, high-precision annotated Uses Cases, using a documentation template. Rather than one artifact, students may have interpreted these as two. In all there were 31 separately graded classroom assignments, albeit refinements of earlier submissions, which could have easily been misperceived as "excessive documentation." In hindsight, it would appear that the number of "deliverables" should have been structured to more closely match the number in the Crystal Clear methodology. Even then, it might make sense to restructure the deliverables into a hierarchy tied to project management with documents clustered by phase (i.e. Planning, Analysis, Design, and Implementation).

- *Adopting Use Cases as the course mechanism for specifying functional requirements necessitated several lecture sessions of training.* Students conversant with OOAD were familiar with Use Case briefs (one-paragraph descriptions of functional requirements), but did not have experience writing fully annotated Use Cases complete with pre- and post-conditions, trigger conditions, actors, scope, process description, extensions, and sub-variations. Students conversant with SA/SD had no previous exposure to the concept of Use Cases, whether brief or fully annotated. To compensate for this knowledge deficit, students were given lectures on writing effective Use Cases and asked to complete several in-class exercises on writing low- and high-precision Annotated Use Cases.

In conclusion, integrating an agile development methodology, such as Crystal Clear, into a Project Management capstone is relatively straightforward, requiring no modifications to the project management process, per se. The common document set required by Clear minimizes developmental anarchy stemming from student's disparate backgrounds in systems analysis and design. The lightweight approach allows students to focus on delivering working systems rather than completing diagrammatic exercises. However, while students recognize the importance of analysis and design documentation in delivering working systems, they still perceive such documentation as "busy work". It appears that even an "ultralight" approach is too heavy for those students who just "want to code."

5. REFERENCES

Abrahamsson, P., O. Salo, J. Ronkainen, and J. Warsta (2002). Agile Software Development Methods: Review and Analysis. Oulu, Finland: VTT Publishing. Retrieved June 15, 2003 from http://www.inf.vtt.fi/pdf/

Agile Development Conference Call for Papers, (n.d.). Retrieved June 12, 2003 from http://www.poppendieck.com/papers/CFP.pdf

Beck, K. (2000). Extreme Programming Explained: Embrace Change. Boston, MA: Addison-Wesley.

Bergin, J. (2000). "Learning the Planning Game: An Extreme Exercise." Retrieved June 13, 2003 from http://oopsla.acm.org/oopsla2k/postconf/Learning_the_Planning_Game.htm

Cockburn, A. (1997). Surviving Object-oriented Projects. Boston, MA: Addison-Wesley.

Cockburn, A. (2001). Agile Software Development. Boston, MA: Addison-Wesley.

Cockburn, A. (n.d.). Crystal Clear: A Human-powered Software Development Methodology for Small Teams *(In Draft).* Retrieved June 12, 2003 from http://alistair.cockburn.us/crystal/books/cc/crystalclear.doc

Davis, G.B., J.T. Gorgone, J.D. Couger, D.L Feinstein, and H.E. Longenecker, Jr. (1997). IS '97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems. New York, NY: Association for Computing Ma-

chinery and Park Ridge, IL: Association of Information Technology Professionals.

Gorgone, J.T., G.B. Davis, J.S. Valacich, H. Topi, D.L. Feinstein, and H.E. Longenecker, Jr. (2002). IS 2002: Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems. Atlanta, GA: Association for Information Systems.

Thomas, S. (n.d.),."An Agile Comparison." Retrieved June 15, 2003 from http://www.balagan.org.uk/work/agile_comparison.htm