Software engineering class eating its own tail

Samuel Mann
Information Technology
Otago Polytechnic, Dunedin NZ
smann@tekotago.ac.nz

Lesley Smith
Information Technology
Otago Polytechnic, Dunedin NZ
lsmith@tekotago.ac.nz

Abstract

This paper describes an experiment where software engineering students were given the task of developing a project management system for use in capstone projects. The Agile Development Framework, which is a combination of agile and structured methodologies, is introduced. Using examples of student work, the paper describes the effect on learning of this recursive approach to learning software engineering.

Keywords: Software engineering projects, computer education

1 Introduction

In this paper we describe the results of an experiment in teaching software engineering. By means of a project based teaching approach, students were given the task of developing software for teaching software engineering.

Teaching software engineering at undergraduate level poses the challenge of presenting a robust discipline to students while reflecting industry currency, as software engineering methodologies have been continuously evolving since inception.

In previous papers we describe the ongoing development of an approach to teaching software engineering (Mann and Smith, 2001, 2004, 2006).

A disadvantage of the project based approach is that students focus excessively on learning the subject matter of the project, rather than the software engineering process that is the objective of the course. The research question for this paper is – can we harness this incidental learning by selecting the development of a project management tool as the class project?

1.1 Context

Software Engineering is a one semester course in the second year in three year degree in Information Technology. It is a compulsory pre-requisite for the capstone project and as such it gives the students the tools for undertaking their projects (Mann and Smith, 2004, 2006).

The course is taught using a project based approach, following a strategy of making it real ("real projects for real clients") and following an "empowerment" approach (Robinson 1994, Smith, Mann and Buissink-Smith 2001). The approach should incorporate a real-life client to mirror an industry experience. It should be flexible (to demonstrate adaptability to change, both in the project and in teaching). It should have a user focus – rather than a plan focussed approach, and it should be applicable over a wide variety of projects, including hardware and network based projects.

Iterations of the course have included a ship safety system, an online motivation project, an animal ethics management system, systems for a maritime museum, a job management system for engineering firms, and a student management system.

A key question in this project based approach to software engineering is the choice of project. In Smith and Mann (2004) we examined our history of such projects and developed a set of guidelines for the selection of the project. In addition to being real, exciting and interesting, we stated that the project should

> *"facilitate teaching the structure of the chosen methodology. For early stage developments the client should have an idea of a business problem, but not a solution…facilitate teaching each of the methodologies' range of tools and techniques. The more creative projects are better for logical design work but are difficult to apply to data modelling". (Smith and Mann, 2004)*

In Mann and Smith (2005) we investigated the relationship between project methodology, the scale of the project, and student learning. The value of a formal methodology in providing a pathway was demonstrated by a large and complex project:

> *"At the onset I had no clue of being able to do what was required, so I didn't have a preconception about what it would look like. I did not think we could do it and definitely not me. Now I see it can be done…"* (Student review)

A potentially huge project, though, can have a disempowering effect, as it was clearly not feasible and students lost interest. Poor groups responded by scoping this project very small, to the extent of developing little more than login systems.

> *"When we first looked at the brief for Captain Black we thought that the scope for the project had the potential to be much larger than anything we could confidently develop"* (student review)

At the other end of the scale, a very small project can further shrink as students lose interest as a result of feeling that a formal approach is overkill. In the mid range are projects that turn out to be much bigger than students' initial understanding.

These examples demonstrate a relationship between the project and the learning of the actual subject: software engineering.

> *"We felt disadvantaged, the other groups had members who had worked in business and knew what the system should do"* (student review)

A disadvantage of the project based approach is that students focus excessively on learning the subject matter of the project, rather than the software engineering process that is the objective of the course.

> *"We had no idea, none of us knew anything about ships, we spent the first few weeks becoming experts on shipping"* (student review)

In the guidelines for the selection of the projects we argued for real, exciting and interesting, and stated that the project should "facilitate teaching the structure of the chosen methodology. Can this facilitation be achieved by the selection of a project management tool as the class project? In focusing student effort on project development, can we generate a deeper understanding of the methodologies and tools involved?

2 Method

Students were given the project of developing a Capstone Project Management System. In a break from our usual practice of using "real" external clients, one of us was the client.

Using a qualitative approach, quotes from student work, reflective journals, etc are given here to identify emergent themes. Two groups (here coded H and C) are used to illustrate contrasting levels of understanding and approaches.

The Capstone projects (and hence Software Engineering) follow an integrated methodology that combines elements of both agile and structured software development. This Agile Development Framework (ADF) approach is described more fully in Mann and Smith (2006).

The focus of the methodology is on the production of robust working systems (software, hardware and maintenance documentation). Planning, comprehensive development documentation and processes are important but are 'means to an end' with a focus on content rather than format/representation.

Noble, Marshall, Marshall, & Biddle (2004) note that the shift to an agile approach in industry "has created a need for a similar shift in software engineering education", explaining that "document centric project methodologies do not align well with students' reasonable expectations of more agile working methods".

Each iteration of the development cycle is divided into "sectors" defined by a deliverable output and communication with the client. The sectors here can be seen to form a structured development process (Figure 1).

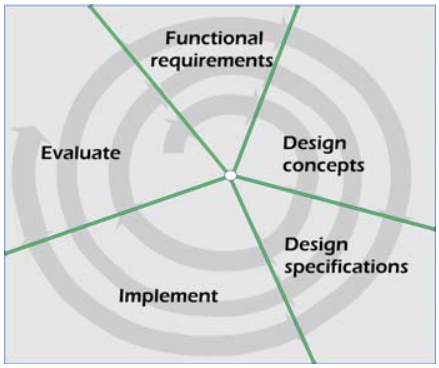

Figure 1 Sectors

The Agile Development Framework comprises three iterations. The first iteration is aimed at building understanding within the development group and stakeholders. The second iteration is aimed at designing and releasing a functional system that meets many of the functional requirements. The third iteration, "robust delivery" is intended to review the success of the second iteration in meeting business requirements, to review functional requirements, and to deliver a robust and stylish "bullet proof" implementation. Each sector is defined by what it produces (Figure 2); the focus is on achieving that outcome. Processes within each sector are determined by agile principles using integrated templates.

The 16 week teaching schedule of software engineering follows the three iterations. In the first four weeks students are introduced to the agile approach and use agile tools to produce a project proposal. The second iteration takes weeks 5-10 and can be considered analogous to a single iteration of a structured approach (with constant reminders of the agile context). Students produce a functional delivery. With little directed structure students take control in the third iteration in working independently as groups to produce a "robust delivery". This third iteration takes weeks 11-16.

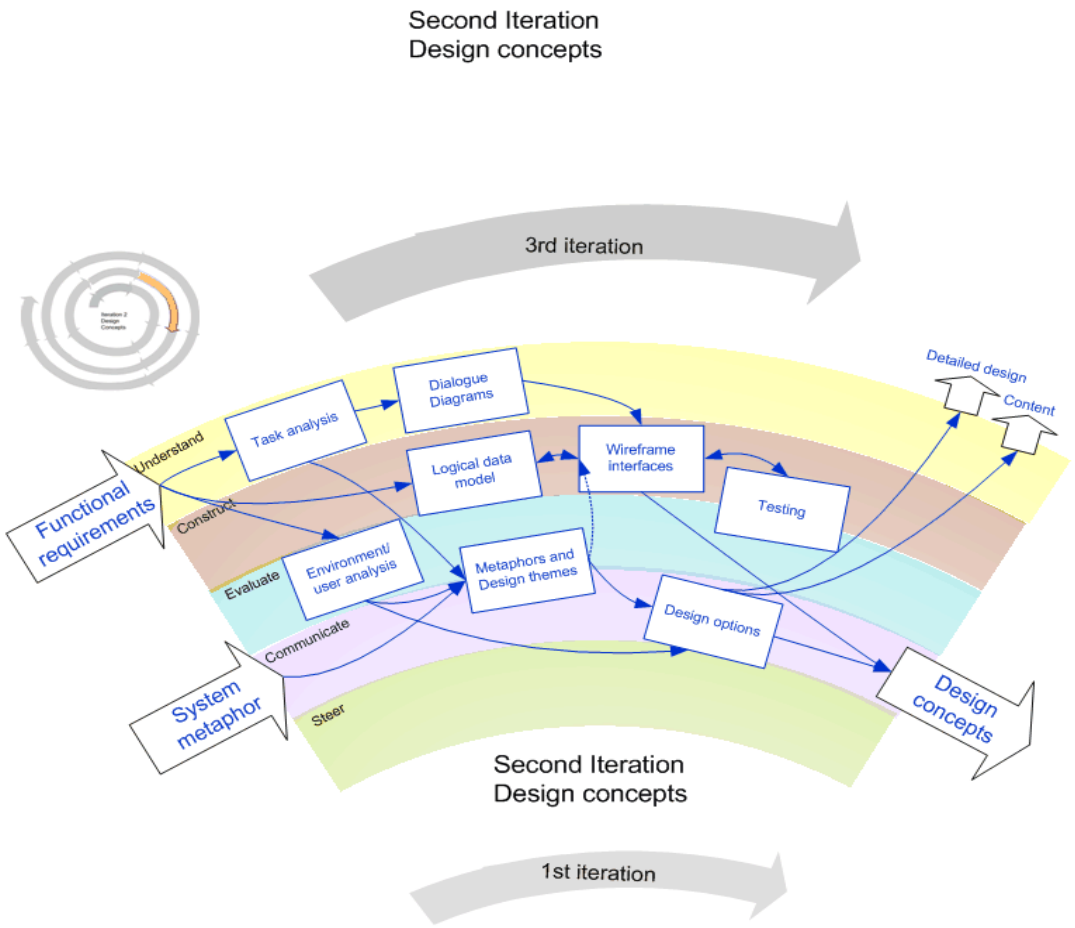

Figure 2: Example sector Design Concepts from 2nd iteration

3 Results

3.1 First iteration

In the project the first iteration is scheduled for two weeks. The intention is that after group forming processes, a few days are spent investigating potential technology and building a very rapid prototype. This serves three purposes, first it is a chance for the newly formed group to work together, second, it helps to identify the business problem (is it something that could be solved with something like this), and third, it gives some insight into the complexity of the technology that is likely to be involved.

The project is introduced to students through a client interview. The student notes (Figure 3) illustrate a tight integration of the project and software engineering – it is not really possible to tell to which these notes refer.

The integration had clear benefits for this group:

> *"material that was collected during our research met two goals. One was to inform ourselves on the components of the Agile methodology and iterative processes; and two was to find examples of project management software that already existed."* (Final review, Group H)

However, the limited understanding of the approach was exposed in the client letter from another group:

> *"we have decided to develop this system with a combination of the Spiral and Scrum Methodologies"* (Client letter, Group C)

A significant part of the first iteration is an attitudinal change by students. Unlike most courses, where assignment requirements are clearly specified, in software engineering, the groups themselves manage the development of the requirements of the projects. Students often have an assumption that we know the functional requirements and are keeping them hidden. A big part of the learning in the first iteration is of responsibility for development. The agile practices of system metaphor and the planning game are useful in this process.

- *Achieve a system that can facilitate people doing their projects*
- *Shouldn't be a disconnection between project and documentation/ management*
- *Make backups regularly, high chance of losing everything*
- *Understand, construct, evaluate, steer and communicate – must remember all five*
- *Group collaboration: need way of allocating work to group members, don't give specific tasks to people, hard to compile at end*
- *System should be supporting agile manifesto*
- *Like to have templates for some things. Ownership of ideas could be a problem*
- *Need a public front end to system, be able to cope with differing amounts of client involvement.*

Figure 3: Student client interview notes (Group A)

For this project the system metaphor of a fridge door was extensively explored in class:

> *For me the project is a fridge door:*
> *always there (you don't have to open it to find information), messages to family (from each other, phone messages), calendar (today, arrangements, upcoming events, shopping list, progress charts, current work, document repository (music tickets, bills to pay), reconfigurable (information held together by fridge magnets), public place, central place: go past it as part of normal life (work)flow (but without it jumping out and interrupting) achievements, photos, newspaper clippings, limited space, information has to be managed to avoid loss in clutter (but this is done without any rules, design or manager) (note, I did write "it does this" here, the door itself of course doesn't do anything except act as a static repository, it is the family who actively manage the information), phone numbers, menu for pizza place, ...and does all these things without getting in way of real job (keeping food cold - our system mustn't get in way of project). (Notes from classroom session)*

Other metaphors explored were: diary, library, mind mapping, job management system (a naïve metaphor).

Some groups used the system metaphor well:

> *dashboard metaphor, forward movement, dials for information, everything within reach, doesn't distract from main function (driving car).* (First iteration, Group H)

This group recognised early on that the measure of successful development was "*better project outcomes and streamlined process"* (First iteration, Group H) while most other groups were seeing non-functional metrics such as error counts as the best way of assessing their success.

One mature student objected strenuously to the time spent on system metaphor and the planning game "let me just talk to users" and was quite frustrated when we pointed out that she was one herself!

Although useful at the start, the confusion over the project and the methodology soon began to hinder some groups:

> *"System Metaphor Description: For this project we will be using the Spiral methodology, incorporating elements of the Iterative Methodologies. The spiral methodology extends the waterfall methodology by introducing prototyping".* (First iteration, Group C)

Some weaker students really struggled to get beyond these early stage tasks. They saw them as major pieces of work rather than small exercises aimed at increasing understanding and communication. Although this is common, it was exacerbated by the close links between the project and the course – while previously we could say "forget ships for a while, let's carry on with software engineering" it was difficult to say "forget software engineering for a while, let's carry on with software engineering".

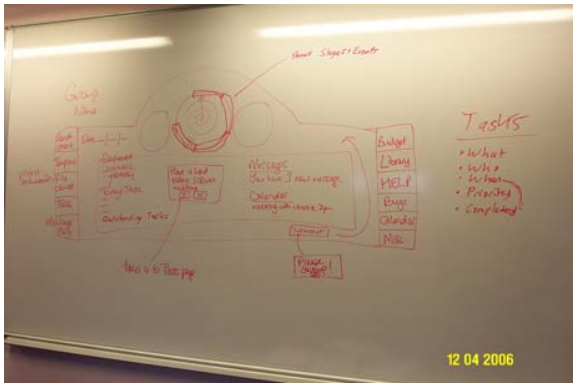

Figure 4 Group H: Iterations combined with dashboard metaphor

For the first iteration in software engineering the deliverable can take almost any form. Group H presented a whiteboard diagram of the direction they saw the project taking (Figure 4). In this they extended their dashboard metaphor and incorporated the iterations of the

ADF, but the content area is sparse with "messages, calendar, click here to go to tasks"

Group C finally came to understand the separation of the project and the course and presented a spreadsheet-based approach that aimed to "*assist students with a better information management system*" (First iteration, Group C*)*. They did not quite understand what the system would do: "*reduce extra costs, user friendly, improve data security*" (First iteration, Group C*)* although the prototype spreadsheets to calculate cost of work did show some creativity.

3.2 Second iteration

The second iteration ends with the development of a functional system. It starts with an analysis stage where groups develop entity relationship and dataflow diagrams (etc) on their way to describing functional requirements.

It is usual practice for student groups to first write meaningless functional requirements. We had hoped that being closer to the project than usual would lessen this problem. It didn't work; the first functional requirements consisted of variations on: "*The system shall store, retrieve and move the project data*" (Group C*)*.

We then had the groups analyse the data form and content. This didn't work either, the weaker groups slipped back into their confusion about project and course.

The breakthrough came by getting the groups to write a job description for a person to be employed to facilitate the capstone projects. Suddenly the level of understanding about both the project and course was raised a great deal. This led to questions based around structures of the ADF (Figure 5).

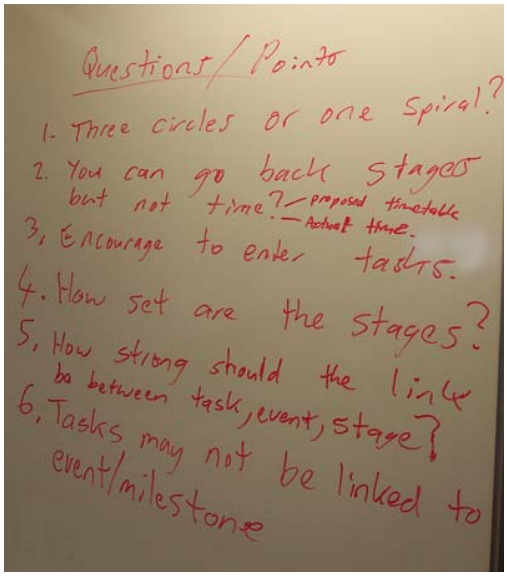

Figure 5: Development questions (Group C)

The functional requirements (admittedly still not perfect) were very much improved:

- *Provide process templates for the student's project throughout the three Iterations: To guide student throughout the process and allow them to check on the progress of the project.*
- *Provide a task management function to set up, choose task and check overall progress of tasks and monitor tasks' progress. Students can set up tasks by enter task name and description. Then enter new taskID into Iteration Sector task table*
- *Enable user to select Iteration and Sector and choose Task To guide student in the process of task management through the use of spiral methodology*

(Second iteration, Group C)

In the second and third iterations, students are expected to work out their own pathway through each sector. Blank "rainbows" are given out for groups to populate with their own workflow, the only requirement being evidence of a rational flow of information between the inputs and outputs of the sector (Figure 6).

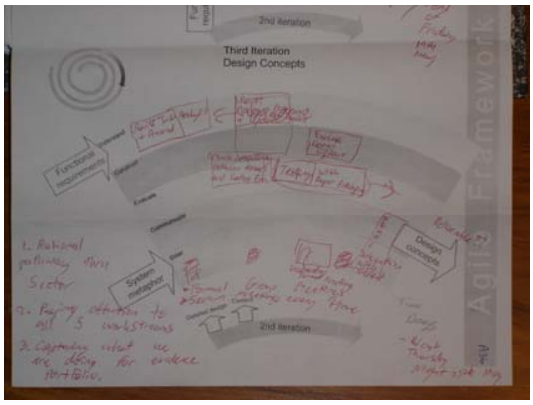

Figure 6: Group working on their own management

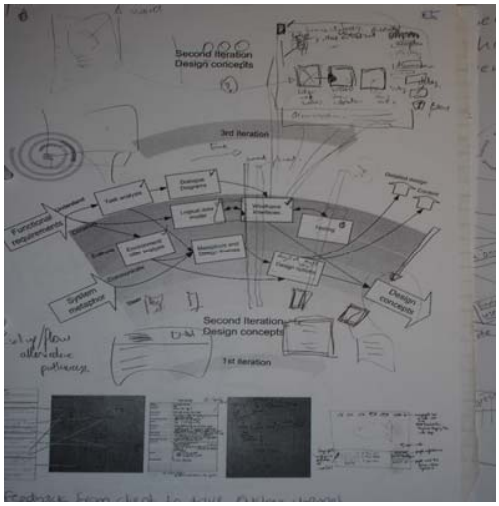

Figure 7: Group management of tasks

Group H saw this flow of information and managing the flexibility of the tasks as the key to the system. They used their own management notes (Figure 6 and Figure 7) to develop a system focused on this support for agility whilst keeping track of the management of tasks. This they developed and tested through paper based prototypes (Figure 8 and Figure 9).

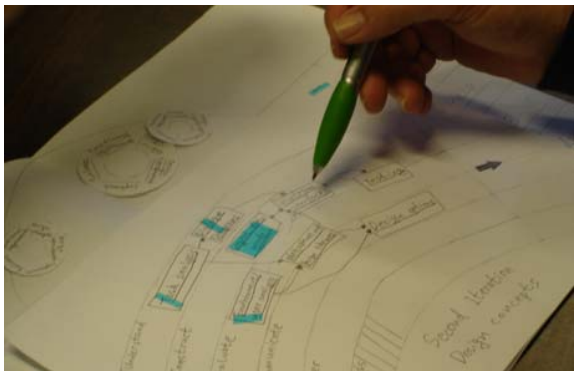

Figure 8: Group H: The content area has changed to represent the structure of the ADF.

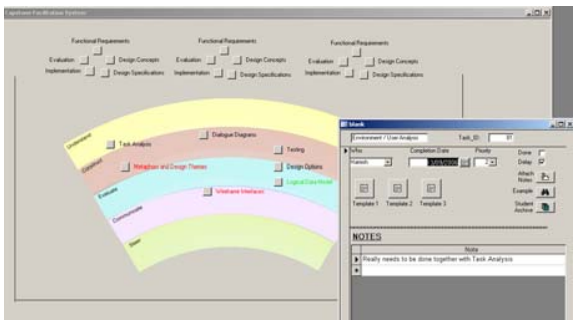

Figure 9 Group H: Second iteration deliverable

"Once a stable database structure was constructed with well-established relationships, we decided to produce a front-end to test the robustness of the back end. Our main goal was to develop the core functionality of the capstone system (the reasoning being that if we could develop the core functions then the other peripheral functions would be relatively easy to incorporate into the system at a later date.

What happened…It worked! We were satisfied with the product and felt confident that it did what the system was designed to do eg assign and prioritise tasks, provide access to templates, allowed the addition of notes, and that it proved the stability of our backend.

We decided the best way to test this release was to use it to complete the project. We found that it required a minimal amount of effort to use and allowed us to track what was happening with each task" (Second iteration, Group H)

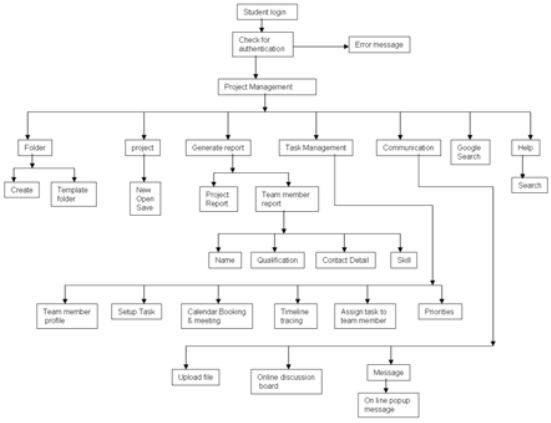

Figure 10: Group C (supposed to be) dialogue diagram

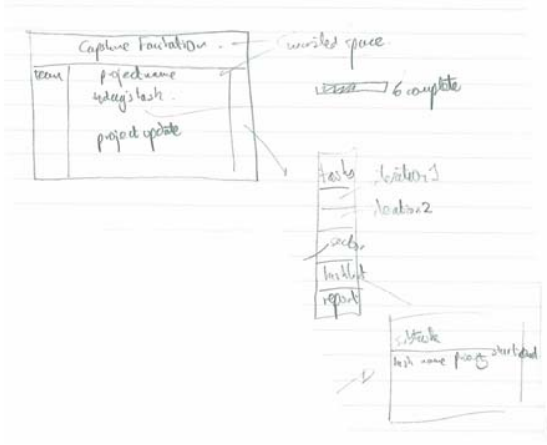

Figure 11: Group C, based at task level and % complete indicator.

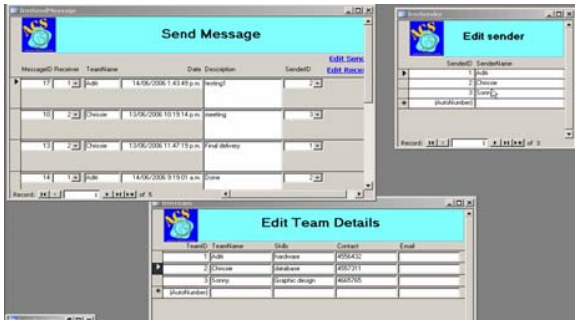

Figure 12: List based functional delivery

While Group H developed what could be considered a holistic or integrated approach (perhaps based on their deeper understanding of the ADF), Group C, on the other

hand, continued to focus on detail. This can be attributed to a weaker understanding of the ADF, and in particular, little understanding of interactivity.

> *"We did a certain amount of prototyping of design in Iteration Two. With the use of Excel to create interface and the cut-out papers and symbols, we were able to test many users for opinions. It gave us a clearer view as how to navigate around the software environment and understand more of the details that needed to be implemented in assigned tasks in the subtask stages".* (Second iteration, Group C)

Figure 10 shows what should have been a dialogue diagram representing interactivity, but instead looks more like a menu structure. This flows through to interface design (Figure 11) and components of the Functional Delivery (Figure 12).

3.3 Third iteration

In the third iteration, groups go around the cycle again, working towards a Robust Delivery. In terms of teaching, we return the focus to agility, instead of structured classes, we spend the time facilitating scrum meetings, steering paired programming and so on.

In the third iteration Group H moved their system to the web (Cold Fusion, Figure 13). The system is essentially the same as the previous iterations. There is a system of templates with edit and later upload. Multiple tasks can be opened.

The group sensibly saw that some features could not be completed in the confines of Software Engineering and would have to wait: this became known as "Iteration 4". This is an appropriate interpretation of our intention in the ADF – timeboxing – delivering what you can within a set amount of time and resources is critical to development (Tate 2006). On reflection, the group recognised that some decisions were inappropriate. The choice of what to develop and what to leave until later was somewhat flawed – they had spent time developing the "cracked dials" rather than fixing the actual function of annotation.

After completion the group also recognised that while promoting flexibility, the system had little to explicitly support agility. Here, the integration of course and project has aided learning.

Figure 13: Group H final system

The final product from Group C looks similar to that from Group H. Figure 14 shows a similar spiral and rainbow as the index but then the lists of tasks are poorly integrated (no advance on Figure 12). The group did demonstrate substantial progress in this iteration but, unlike Group H, the change appears to be forced, rather than based on a deeper understanding.

Changes were made somewhat grudgingly in response to "client" instruction but this wasn't mirrored in general understanding. On reflection the group stated:

> *"We did have lots of changes requested from the client; we followed the agile process of using feedback to further improve our design decisions."* (Third iteration, Group C)

But, while accepting the agile approach of embracing change, discussion about the fundamental purpose of the system (and by association the ADF) was displaced by lots of mechanical details. Client interaction took the form of lots of detailed questions:

1. *Can Joe display all tasks on one screen?*
2. *Is the system automatically saved on a regular basis?*
3. *Does the system remind Joe that he needs to backup his project?*
4. *Can Mary view a list of previous reports printed to avoid reprint?*
5. *Can Mary also access the system to find out who has printed what reports?*
6. *Can the system handle all iteration on one screen for a print out?*

There are fundamental errors in design, for example, the percent complete bars for each iteration (cf the required timebox approach). This reflects how far behind this group was (although this tardy development would have happened even if a different project had been used).

Again, there was little support for agile concepts, the system supporting less flexibility than Group H and, with tasks needing to be assigned to only one person, the system actively discourages teamwork and paired programming. In the final presentation, this group described their system as *"providing support for the spiral methodology*" rather than Agile Framework.

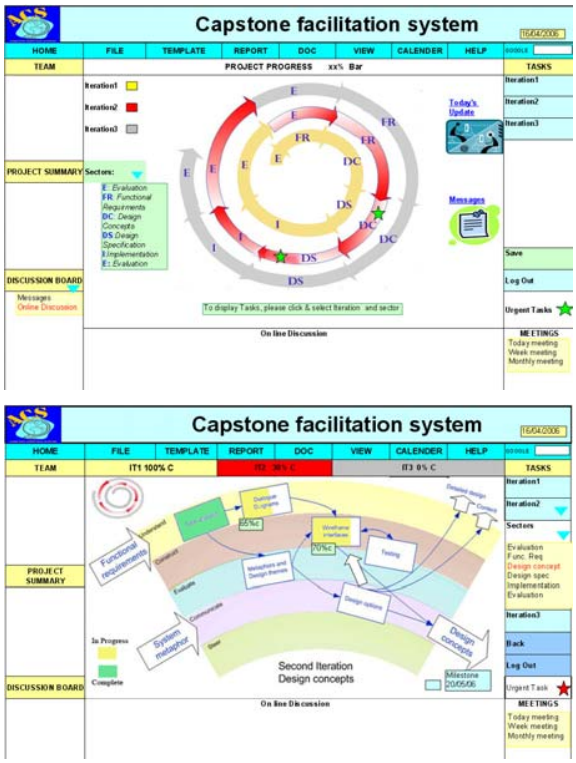

Figure 14 Group C final system

All students in Group C and H passed the project component of software engineering, the exam and the course.

4 Reflection

In this paper we have described an attempt to leverage the incidental learning associated with the project to assist with learning of software engineering material. We did this by using the development of a support system for Capstone projects as the project for software engineering. The questions now are: did it work and would we do it again?

On balance, we conclude that it did work. The trade-off is between a potential confusion of the course and the project, and the benefits of "double dipping" on learning.

> *"Overall this has been a worthwhile experience for me technically and personally, in learning the process of software engineering, and in developing skills to become more adaptable to the ever-changing work environment."* (Student review, Group H), and

> *"In my final conclusion of this project I would like to reiterate that this was an enjoyable learning experience for me. As I gained more understanding of software engineering I also develop a greater excitement and passion for the IT sector. So many times in class I was distracted by new software ideas going through my head and having to write them down I sometimes missed what was being taught in class."* (Student review, Group C)

Two major areas need addressing: formalising client interaction and requirements of implementation.

We had hoped that having one of the lecturers act as the client would mirror the extensive client involvement promoted by agile approaches (indeed this has always been a difficulty of "real clients"). This didn't work effectively, instead it added to student confusion:

> *"We feel that the client interaction was too informal during this process; however this was probably due to ill-defined boundaries where the client was also the lecturer and also the mentor."* (Student review, Group C)

In future work, we hope to formalise this interaction.

A second area of concern is the requirement to implement systems as part of software engineering. Before we adopted the ADF, the course was document centric and, as the project was not implemented, students were moving into the capstone project with little awareness of implementation issues and often failed to produce the required outcomes.

Getting students to do implementation can have the effect of restricting their design thinking – design decisions are made on the basis of limited technical ability eg: user look up rather than linking. Crucially, however, in this double dipping approach, limited design can be closely related to limited learning.

Strong groups can see past this difficulty:

> *"We also feel that since this is our first time directing a software engineering project with the focus on learning rather than on getting every step correct, that our group cannot be too critical of our effort and overall, we consider we have achieved a successful project for the client and the end users"* (Student review, Group H)

> *"By following an iterative approach, teams are not focused on producing code, but instead are able to focus on project innovation. This focus increases innovation and cuts delivery*

time and also encourages parallel-development activities." (Student review, Group H)

But for weaker groups this can become a barrier to learning (although it provided many opportunities for learning about conflict resolution and group dynamics).

"There is lots of paper work to be done; however the detailed analysis steps helped us to understand the process of the project clearly. I think that the relationship between what we originally conceived as our project, and what was our final outcome, was very close. Next time around I would like to prepare myself better in programming so that I can manage to design the software more efficiently." (Student review, Group C).

"As we are all new to this process, we had neither the ideas nor the experience to anticipate the difficulties of working together under a high-stress load. Secondly, with open communication, understanding of the individual commitments and capabilities will certainly help in establishing a functional group able to work together." (Student review, Group C)

5 Conclusion

The use of a project management system as the project for the software engineering class led to some confusion for some students. However, the benefits of the approach were clear for those students who were able to effectively move between the dual roles of developer and user. The question "what would the user need now?" was easily addressed, and generated useful group discussions.

Apart from the difficulty of using a lecturer as client, the experiment was worthwhile, especially in promoting useful class discussion around the critical area of user requirements. The students' awareness of project management was extended beyond what they would have gained through a traditional project.

6 References

Mann, S., & Smith, L.G. (2004) Role of the development methodology and prototyping within capstone projects . *Proceedings 17th Annual NACCQ*, Mann, S. & Clear, T. (eds). Christchurch. July 6-9th 2004. p119-128.

Mann S., & Smith, L.G. (2005) Technical complexity of projects in software engineering *Proceedings 18th Annual NACCQ*, Mann, S. & Clear, T. (eds). Tauranga. July 10-13th July 2005. p249-254

Mann, S. and Smith, L.G. (2006). Arriving at an agile framework for teaching software engineering. 19th *Annual Conference of the National Advisory Committee on Computing Qualifications*, Wellington, New Zealand, NACCQ in cooperation with ACM SIGCSE. 183-190

Robinson, H. (1994) The Ethnography of Empowerment: The Transformative Power of Classroom Interaction The Falmer Press.; Bristol:

Smith, L.,G., & Mann, S. (2004) Projecting Projects: Choosing Software Engineering Projects, Proceedings 17th Annual NACCQ, Mann, S. & Clear, T. (eds). Christchurch. July 6-9th 2004. p183-190.

Smith, L.,G., Mann, S., & Buissink-Smith, N. (2001). "Crashing a bus full of empowered software engineering students." New Zealand Journal of Applied Computing and Information Technology 5(2): 69-74.

Tate, K. (2005). Sustainable Software Development: An Agile Perspective, Addison Wesley Professional, NY. 264p