Rescue of a whole-farm system: crystal clear in action

Pablo Mangudo* and Mauricio Arroqui

Universidad Nacional del Centro de la Provincia de Buenos Aires
(UNCPBA), Campus Universitario (CP 7000),
Tandil – Bs. As., Argentina
E-mail: pmangudo@gmail.com
E-mail: mauriarroqui@gmail.com
*Corresponding author

Claudia Marcos

ISISTAN Research Institute,
Facultad de Ciencias Exactas,
Universidad Nacional del Centro de la Provincia de Buenos Aires
(UNCPBA), Campus Universitario (CP 7000),
Tandil – Bs. As., Argentina
E-mail: cmarcos@exa.unicen.edu.ar

Claudio F. Machado

Universidad Nacional del Centro de la Provincia de Buenos Aires
(UNCPBA), Campus Universitario (CP 7000),
Tandil – Bs. As., Argentina
E-mail: cmachado@vet.unicen.edu.ar

Abstract: This paper presents a case study of an agricultural project. From a desktop research model (stage 1), a web-based whole-farm simulator was developed applying a waterfall life cycle (stage 2) but several problems were detected and the project failed. The project was continued (stage 3) applying crystal clear agile method, which suited better the requirements. An efficient team communication and the frequent delivery of usable code increasingly contributed to the sponsor's satisfaction. It was positively concluded that crystal clear was able to rescue the project and that it could be applied in a short-term period without major difficulties.

Keywords: agile method; AM; agricultural system; study case; experience.

Reference to this paper should be made as follows: Mangudo, P., Arroqui, M., Marcos, C. and Machado, C.F. (2012) 'Rescue of a whole-farm system: crystal clear in action', *Int. J. Agile and Extreme Software Development*, Vol. 1, No. 1, pp.6–22.

Biographical notes: Pablo Mangudo is a System Engineer, graduated on 2008 at the Faculty of Exact Sciences of the Nacional University of the Centre of Buenos Aires Province (UNCPBA). This paper reports his shared undergraduate thesis with Mauricio Arroqui. At present, he is contracted by the project, and also a Software Entrepreneur in the agricultural domain, through a FONSOFT Project (http://www.agencia.mincyt.gov.ar).

Mauricio Arroqui is a System Engineer, graduated on 2008 at the Faculty of Exact Sciences of the Nacional University of the Centre of Buenos Aires Province (UNCPBA). This paper reports his shared undergraduate thesis with Pablo Mangudo. At present, he is contracted by the project, and also a Software Entrepreneur in the agricultural domain, through a FONSOFT Project (http://www.agencia.mincyt.gov.ar).

Claudia Marcos is a Senior Lecturer at the Faculty of Exact Sciences of the Nacional University of the Centre of Buenos Aires Province (UNCPBA). She is a Researcher at Comisión de Investigación Científica de la Provincia de Buenos Aires (CIC). She graduated as a System Engineer on 1993 and obtained her PhD on 2001 in the same institution. Her main research interests are aspect-oriented development, UML and agile methods.

Claudio F. Machado is a Senior Lecturer at the Faculty of Veterinary Sciences of the Nacional University of the Centre of Buenos Aires Province (UNCPBA). He graduated as a Veterinarian on 1990 in the same institution. He obtained his MSc in Animal Production on 1993 at University of Chile, and PhD in Animal Science on 2004 from Massey University New Zealand. Actually, he is leading different project associated to simulation of agricultural systems.

1 Introduction

Software as a research support tool is being increasingly used in different domains and agricultural systems are not the exception. Pastoral agriculture occupies around 20% of the land surface of the globe, and is directly or indirectly responsible for meeting the economic and material needs of a substantial proportion of its human population (Illius and Hodgson, 1996). In Argentina, ruminant productive systems are primarily under grazing conditions, which represents the cheapest source of available nutrients for ruminant production, and in consequence, the greater the control the livestock producer exerts over forage production, consumption and matching animal requirements to seasonal forage production cycles, the better are the chances that the operation will be profitable (Forbes, 1988).

Agricultural systems are complex (Pearson, 1997) as their different components (climate, land, pasture, animal intake, animal growth, market, management, etc.) interact in time (Pannell, 1999). Consequently, in order to gain insights about the whole system, simulation has been used to study such systems (McCall et al., 1994) and decision support systems (DSSs) for whole-farms have been developed for different conditions from those Argentina's (Doyle et al., 1989; Loewer, 1998). Based on some local research models (Berger et al., 2002; Machado, 2004) a project (PICTO 22926, 2006) was carried out to develop a web-based DSS by inclusion of existing or extended procedural modules following initially a waterfall life cycle. Due to increasing problems in the project development process, it was submitted to a revision. This article presents the experience case of such a whole-farm simulator project. The paper is structured as follows: Section 2 presents the development stages of the simulator, including detected problems. Section 3 describes the application to the simulator of a particular AM, called crystal clear (CC) and some properties of the methodology are summarised. Section 4 shows an assessment

of the system and the first four iterations. Lesson learned and conclusions are presented in Section 5.

2 Stages of a grazed-based beef cattle simulator

The development of this simulator can be divided into three stages. At stage 1, the project was initiated (2002–2004) using the modular-based simulation shell provided by a fourth generation language, 5.0 Extend (Krahl, 2002). Extend™ had been used previously to develop whole-farm models (Brennan and Gooday, 1998). This stage was successful regarding the initial research objectives, which were basically to develop a simple simulation research tool directly by the expert domain to gain further insights about pastoral systems (Machado, 2004).

On February 2005 the stage 2 of the project started when it was decided to increase system usability for educational purposes. The Extend™ simulator was licensed, too sophisticated and poorly intuitive for novice users; hence, different technologies were selected and incorporated. A web-based system, with its views and outcomes divided by user profile (e.g., researcher, advanced student, etc.) were considered for the selection of those technologies. A team including a project manager and three developers with a weekly workload of 30-hours per member using a waterfall life cycle was organised for stage 2. Software development was carried out all over 2005 and 2006, but different increasing problems threatened the project continuity. The team had difficulties to follow defined plans and the client was unsatisfied with the progress of the system. The following main (and related) deficiencies were detected:

- Deficient communication. Expert domain (and sponsor) was present only six-hours per week. Telephone and e-mail were also used, but these means did not result efficient.
- Developers were novices on the domain. Associated to the previous point, this caused some tasks were too partitioned and not adequately integrated to the whole system requirement.
- Requirements resulted highly dynamic. Therefore, most of the tasks required modifications and re-implementations, causing continuous delays to the development process.
- Deficient documentation of both, the development process and available functionalities.
- No working software was available.

The objectives of the stage 2 of the project were clearly not met. After a meeting between the team and the sponsor, the last decided to continue the project but with adjustments to downgrade possible failure risks. The team requested access to the expert user on a daily basis. On January 2007, the stage 3 of the software development finally started. The team was partially renewed (two developers were contracted by other project, and a new developer was incorporated). Based on the previous experience on the domain and the system, the team worked out a requirement list to identify a proper development methodology for this stage 3, which might be able to:

- cope with changing requirements
- be oriented to small teams
- privilege an efficient communication within the team and client/domain experts
- frequently deliver of usable code to users
- develop a sound documentation of the development process.

Based on the previous list and available options, it was decided to use an iterative and incremental life cycle, as is the case of agile methods (AM) (Larman, 2004). Within them, particularly CC was applied (Cockburn, 2004).

3 Application of CC agile method to the simulator

CC is based on a development team with a leader and several developers (two to seven), in a close seating and direct communication environment, using design sketches, notes and screen drafts (Cockburn, 2004). Frequent testing and deliveries of running versions of the system are also key points of the method. At first, the project required to define short-term objectives to sort out those functionality deficiencies described previously, thereafter to progress to others long-term objectives. CC is based on fixed iterations, therefore, it fitted to the project characteristics and was efficient with requirement changes as new functionalities were available. The application of the different practices of CC to the simulator is summarised below.

3.1 Roles and development environment

The system development team was integrated by four people with the following roles:

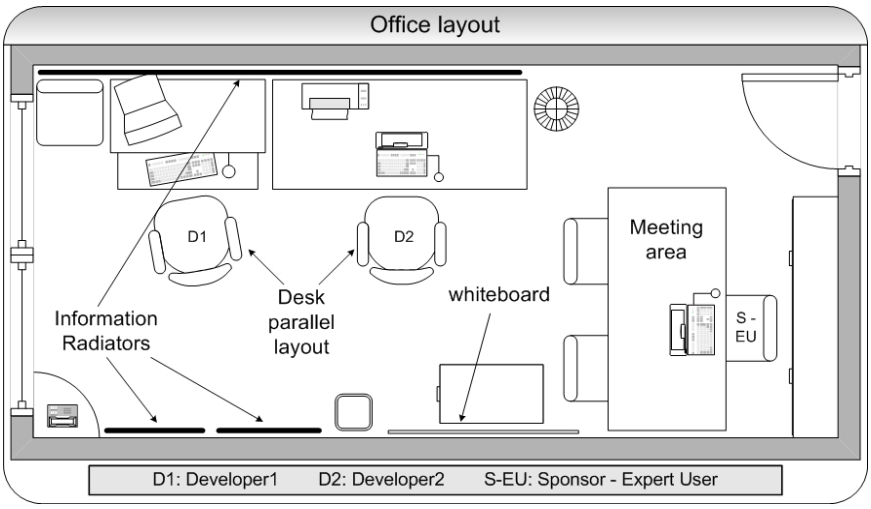
- *Developers*: within this role, CC defines specific categories lead designer, designer-programmers, coordinator, tester and writer. In this case, these roles were played by only two people; hence, they were allocated dynamically depending on particular needs of the running iterations.
- *Sponsor*: administrates the project budget.
- *Expert user*: a domain expert with expertise in the use of the system.

In this project, the roles of sponsor and expert user were represented by the same person. Research funding was provided by a project of the Argentinean National Agency of Research (PICTO 22926, 2006) leaded by the sponsor. He has a vast expertise on domain and domain modelling with fourth generation languages (as the used during stage 1).

- *Software technology consultant*: a computing domain expert familiarised with current available technology and the project who helped developers when required. The former during stage 2 manager played this role.
- *Friendly user*: domain experienced people, who can evacuate doubts or give a rapid opinion as required by developers. Different users carried out this role during the development depending on the issue under review.

The workplace was a 21 m^2 office room, and its layout was changed by the development team in order to match better some of the CC properties (Figure 1). The new arrangement established two well-defined sectors, a meeting area and desk/computer area for developers. Information radiators were in front and back of developers, which were highly visible within the development environment. From stage 3 of the project, the expert user/sponsor shared the office room daily.

Figure 1 Office layout



3.2 Life cycle and properties

AM (including CC) use iterative and incremental life cycle. Iterations are organised in a sequence to achieve an *Incremental Rearchitecture* strategy. Time horizon is fixed for each iteration and defined at its planning time (*time-boxing* practice). Therefore, sometimes it is necessary to suppress part of a planned functionality to keep the time-schedule. Similarly, planned tasks are stable, therefore, new requirements are included in future iterations (d*o not add to Iteration* practice). When an iteration is finished, a complete analysis of its outcomes is required in the shape of a *reflection workshop*, involving the whole team. These workshops allow improving the development by the technique *methodology shaping*.

Different practices (strategies and techniques) are collected to promote core properties of CC: *frequent delivery*, *osmotic communication* and *reflective improvement*. These properties ensured a safety zone for the projects were achieved in the following way:

- *Frequent delivery*: CC suggests a software delivery interval no longer than three months. Iterations do not necessarily include working code as output (its convenience is evaluated at each iteration planning); however, at least an internal release for the team is usually advised. Planned tasks during an iteration are stable (*do not add to iteration*) therefore, new requirements are left for next iterations. This allowed to keep focus, avoiding reprogramming needs (as in the case of stage 2 of the project) and their inefficient consequences. Additional sources of information for

the sponsor were the *iterations plans*, represented by Gantt charts using Open Workbench tool (2009).

- *Osmotic communication*: CC takes advantage of small team size and proximity to strengthen close communication into the more powerful 'osmotic' communication property. The new office layout (Figure 1) helped team interaction and cooperation by improving knowledge transfer. The practice *side by side programming* was successfully applied. *Information radiators* like whiteboards and flipcharts on the wall, design sketches and notes (like *story cards* from XP agile method (Beck, 2000)) contributed positively to this property and to the awareness of the project progress. In order to avoid unplanned interruptions, the cone of silence practice was also used.
- *Reflective improvement*: This property aims to identify what is and is not working, and then selecting and applying best corrective actions. Reflection may involve diverse topics, like office layout, a new work-product, a new development habit, etc. Reflection may be formal, as the case of *reflection workshop*. Under the shape of this property, changes in technologies, definition of coding standard and other required conventions were part of the activities carried out by the team. This property was very important particularly during methodology training. Additional practices were gradually selected and applied by developers (*self-directed and self-organised*) according different to project needs (*methodology shaping*).

Additionally, CC highlights other properties that may decrease the failure risk of a project. They reinforce CC core properties:

- *Personal safety*: Some of CC properties are based on the fact that the team shared a close and continuous working environment. Both developers had worked together previously therefore they knew their skills and limitations.
- *Focus*: This property highlights the importance of keeping direction and priorities for any programmed task. The work-product *mission statement* (developer by the sponsor) resulted very important to this property, as it contains vision, mission and strategic objectives for the project.
- *Easy access to expert users*: As its name indicates, it means a direct and quick access to domain experts, who facilitate the topic understanding of the developers. From stage 3, expert users shared the office room (Figure 1) on a daily basis.
- *Technical environment with automated tests*: The applications of such tests are oriented to carry out integral tests of the system under development. The team disregarded such tests based on the limitating human resources (only two developers) combined with the need of quick results. These tests were replaced by manual alternatives run by each developer and analysed with the help of the expert user. Additionally, friendly users carried out complementary analysis. *Configuration management and frequent integration tools*: these tools allow developers to work individually but then to integrate her/his activities without major difficulties. Similarly, this property allows for changes in code to be tracked through software versions. In the present project, SmartCVS Foundation (2009) was used with this purpose. Additionally, this tool was used to monitor the versions of document artefacts

Additionally, selected practices suggested by other AM, as *block gone in one day*, *self-directed and self-organised*, *do not add to iteration* by Scrum (Schwaber and Beeddle, 2002), *use-case driven* of UP (Larman, 2004) were also included.

4 Assessing the system and working iterations

At the beginning of the stage 3 of the project, an internal assessment was developed following the *Exploratory 360°* practice, emphasising on the functionality and the architecture of the system. Later, different working iterations were planned and developed.

4.1 Assessing the starting state of the system

System documentation was poor and outdated at the end of stage 2, therefore, initial efforts were oriented to code review by three weeks (two-people, six-hours per day). This activity allowed the adjustment and completion of the software requirements with the active participation of the sponsor. A first *use case and requirement file document* with most relevant requirements of the system was developed. Its content may be summed up as:

> "System should be able to run 'whole-farm' simulations of mixed agricultural enterprises from Buenos Aires Province – Argentina, allowing the evaluation of alternative strategies at different time scales."

It means a dynamic representation (one-day step) of pasture-based beef cattle finishing production and cash crop operations. Pertinent information from the stage 1 of the project (Machado, 2004) constituted major rationale and procedures from domain knowledge. Simulation scenarios should be formally represented, and the system must be controlled by management rules (e.g., animal sales, animal purchases, strategic feeding supplementation, strategic pasture conservation of seasonal surplus, etc.). The simulator should be web-based by authorised users, and views and outcomes of the system should be divided by user profile (e.g., researcher, advanced student, etc.).

Within the simulator different sub-systems interact (Figure 2). All knowledge-related to animal biology (feed-intake, animal growth) and pasture characteristics were contained in the biophysical sub-system. Additional sub-systems of the simulator were oriented to financial, economic and tax issues. Parameters entering to sub-systems might be deterministic or stochastic.

The work-product architecture description was also developed during the system assessment. The simulator was based on three-tier architecture (Bass et al., 1998; Shaw and Garlan, 1996) as shown in Figure 3. The presentation layer was developed in OpenLaszlo (2009). During the stage 2, a separate developer, different from than those in charge of the application layer carried out its development. As some deficits were detected in the presentation layer, it was decided to contract an external team to update it and to make it functional.

Figure 2 Sub-systems within the whole-farm simulator

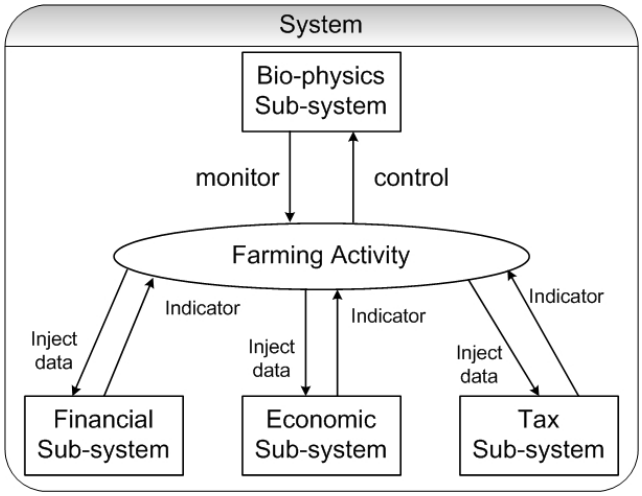
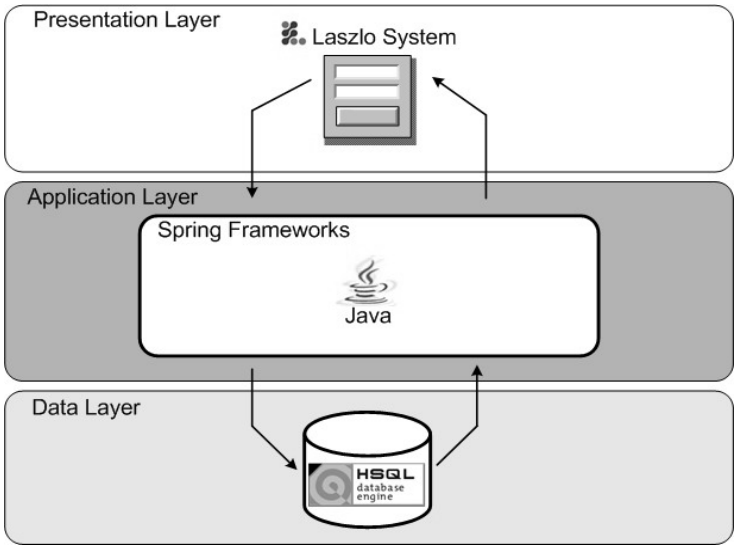


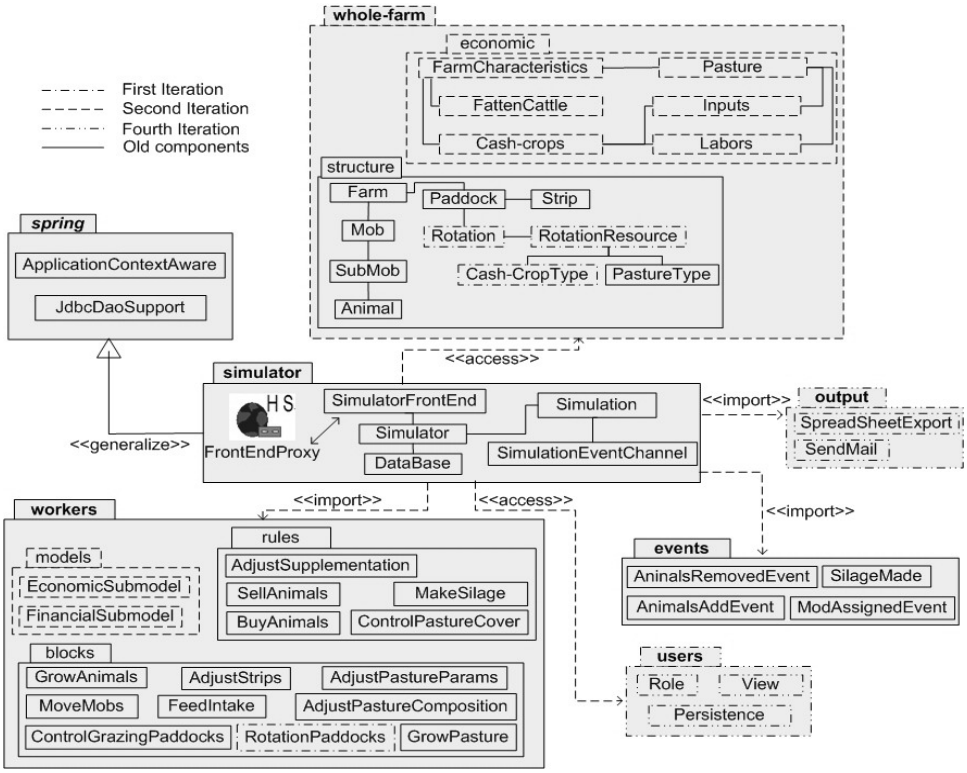
Figure 3 Three tyre architecture of whole-farm simulator



The application layer (Figure 3) is shown expanded in Figure 4. It used framework Spring (2009) to handle simultaneous simulation runs and as an interface to database. A simulation contains the whole information of a farm (paddocks, animals, etc.), which is processed by classes denominated 'workers'. These workers are in charge of different tasks of the farm, as to estimate pasture growth (GrowPasture), animal intake (FeedIntake), animal growth (GrowAnimals), etc. Each worker is not active at each simulation step, but the activity of a worker may require others. Furthermore, an event driven architecture was applied. When a simulation is executed, the system presents alternative internal states as starting, running, finishing or aborting. When the system

detects an anomalous number, aborting options allow feedback to the user, facilitating a quick identification of inadequate simulation parameters.

Figure 4 Package diagram of old and new components



Note: Different lines indicate tasks by iterations.

The data layer utilised Hsqldb database (2009), which was set up to run directly on server RAM memory. Database was prepared to cope with simultaneous simulation runs by a queue arrangement to store the information of each simulation.

4.2 Iterations developing

Once the domain and system was understood adequately, the first task was oriented to get the whole team familiarised with CC. The second work-product *mission statement* was developed by then, where the goal and main characteristics of the development were delineated. It was stated as:

> "A beef cattle system research group of a public university with the following objectives:
>
> a to transfer domain knowledge to researchers, students and private consultants by using specially designed and developed decision support system (DSS)
>
> b to stimulate the use of DSS to increase efficacy and efficiency of decision making process of commercial farms."

Next, work-products were *team structure and conventions* and r*isk list*. In the former, role descriptions, assigned responsibilities to team members and initial conventions for system development were stated. *Risk list* was particularly considered for planning purposes, establishing strategies to decrease identified risks. With these work-products the team was ready to incorporate new functionality to the simulator by means of several iterations (Figure 4).

4.2.1 Iteration 1

By following an *early victory* strategy, a short first iteration (19 days) was designed (22/01/2007 up to 09/02/2007). The transformation of the beef cattle farm to a cash crop-beef cattle farm was planned. It was successfully achieved by adding a rotational use of paddocks, managed by user designed schedules applied to paddocks. During the *reflection workshop* carried out at the end of the present iteration, different code *conventions* were specified to facilitate maintenance and readability of the code. The first integration of the presentation layer (originally developed at stage 2 of the project but updated by a hired team at the beginning of stage 3) with the application layer was planned and achieved as a *working software delivery*. However, the sponsor disapproved the visual quality of the presentation layers developed by the external team. As a consequence, the development team included the point in the *risk list* and decided to develop by itself all required interfaces.

4.2.2 Iteration 2

The second iteration started on 12/02/2007 and ended by 23/03/2007. An economic model was incorporated, allowing estimation of gross margins for cash crops and beef cattle production, asset profitability, net utility, growth capability of the enterprise etc. Training on Open Laszlo (as the selected technology for the presentation layer), was also planned and achieved. On this iteration, the expert user and friendly users also tested interfaces to identify required improvements. As the team increased its system knowledge, a preliminary *project map* (without specific dates), including an activity prioritisation for future iterations, was developed.

During the *reflection workshop* was detected that code integration resulted slower than planned. It was usually developed once a week, but it was changed to an interval of two days. Design, particularly documentation and implementation of the economics model, resulted slower that planned. This point was added to the convention list, limiting project documentation to an hour per day. Furthermore, *project map* and a *release plan* (software delivery) were left for a future iteration.

4.2.3 Iteration 3

This iteration was carried out from 26/03/2007 up to 27/04/2007. Two main activities were planned, both linked to the presentation layer. Interfaces for the economic model developed in the previous iteration, were coded. They also did required improvements to interfaces programmed by the contracted team. The simulator represents a complex domain, so it is important that the simulator was as friendly as possible (included in the *risk list*). Therefore, to achieve that, interfaces play a key role, hence, special care on interface review, planning and development was taken during this iteration. The time

allocation to interface activities resulted lightly underestimated (interface design resulted particularly time-demanding), hence, some components required simplification to achieve the planned due date. No software delivery was planned, but the sponsor was well informed about progress favoured by his double role (expert user). During the *reflection workshop*, it was highlighted the need of a careful timeframe estimation when planned tasks included presentation layer.

4.2.4 Iteration 4

The fourth iteration started on 30/04/2007 and finished on 15/06/2007. By then, team conventions were well understood and in a continuous improvement, favouring a high team motivation. Preliminary *project map* was adjusted to yield the initial release plan. The delivery of a running version of the system accessed by a browser was the main task of this iteration. To achieve this objective, database was migrated from Hsqldb (2009) to Oracle XE (2009). A module was developed to handle user profiles, which defines allowed activities, required inputs, views and outcomes by profile. On this iteration, it was also decided that outcomes of the system were sent directly to the user's e-mail account as a spreadsheet including plain data and pre-designed graphs. A training period to investigate the use of an application program interface for this functionality was carried out. Activities mentioned previously are shown in Figure 5 as part of the *iteration plan*. Along the iterations, the progress (*iteration status*) was showed like *burn chart* over Gantt charts. Also *story cards* (Beck, 2000) were regularly used in the iterations (Figure 6).

Figure 5 A Gantt chart of the fourth iteration

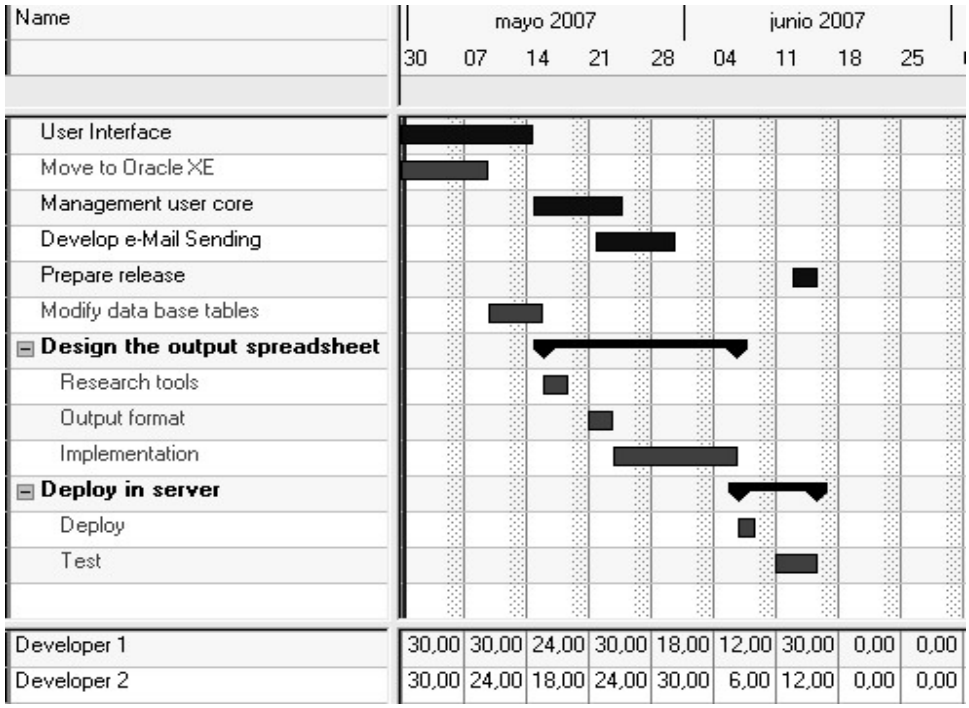


Figure 6 Story cards for user profile management and linked sketches

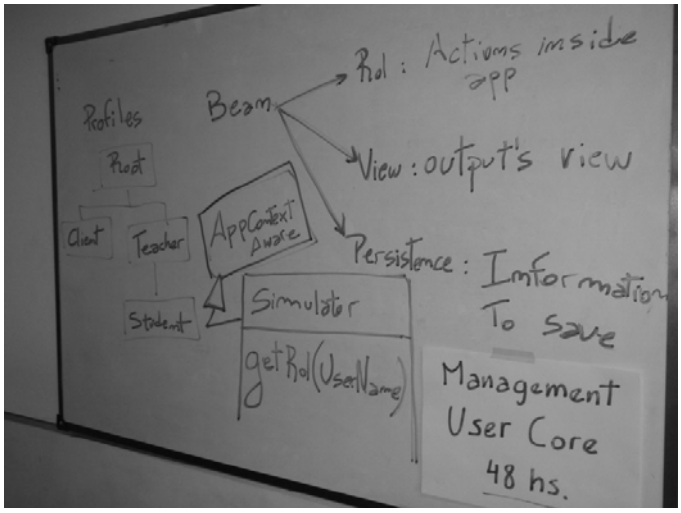
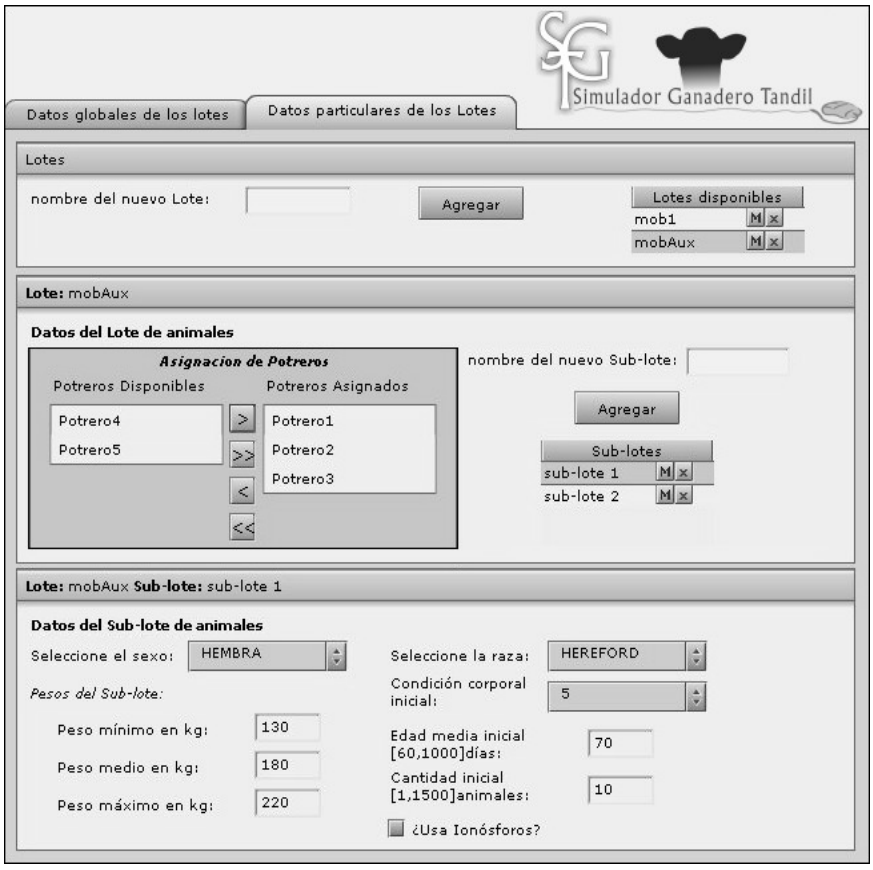


Figure 7 Part of the presentation layer of the whole-farm simulator



User management was added to the simulator package (Figure 4), and a user package was created. Output package was also added in order to yield outputs and to abort a simulation. Automated tests were not used, so each developer at each task carried out her/his own functional tests. System architecture was modified after four iterations. Structure packages and economic packages were incorporated to whole-farm and a worker package was added to model package.

The presentation layer was highly improved and the simulator was seen with a browser. Figure 7 shows a screen to define animal details to setup a simulating scenario. The language used for the interfaces is Spanish because this development is initially for Argentinean conditions. A standard format for interfaces was defined with the expert user and friendly users. It was incorporated as a 'friendly' criterion for all following developments, although further usability tests were continuously carried out.

During this *reflection workshop* no new suggestion were included as the development methodology reached a desired, therefore was highly appreciated by developers and the sponsor.

5 Final remarks and conclusions

Deficits in requirement gathering of the system (Section 2) caused to miss the needed features and to apply an inadequate development process during stage 2 of the project. In the software industry, more than half of the software projects fail to match the required functionalities (Standish Group's The Chaos Report, 2004). However, far from conformity, this paper aims to contribute with an understanding of the initial development failure in a small project, identifying learning lessons which may contribute to other agricultural systems and other complex domains. Cost overruns were an important side effect of the detected problems, as usually reported in the industry (Masticola, 2007). Simulation was only part of the activities of this small research-oriented agricultural development. The discouragement of the sponsor (without any working software available at the end of stage 2) associated to cost overruns, almost cancelled activities before project completion.

At the beginning of stage 2, several meetings between the team and the expert user/sponsor were carried out. A waterfall life cycle was applied and requirement gathering seemed to be well defined. However, awareness of constantly changing requirement arose during the development, leading to project delays. Problems at requirement capture affects different processes like software architecting (Ferrari and Madhavji, 2008), selected technology, life cycle selection, and implementation process, etc. (Thayer et al., 1997).

In this study case, different reasons were likely to contribute to poor requirement capture, hence, to choose a waterfall life cycle, which finally resulted inadequate to the project. At first, the complex nature of agriculture domain where multiple disciplines interact (Pearson, 1997). Secondly, from the authors' knowledge, this web-based whole-farm simulator is the first experience in Argentina, so no local benchmarking was available. Lately, it was reported that AM were applied to a huge agricultural simulator from Australia (Holzworth et al., 2006).

Besides, the previously mentioned problems potentially linked to requirement capture, there were additional difficulties during software development. Some authors (Johnson and Holloway, 2006) have stated that it is often difficult to distinguish between failures in requirement engineering and problems elsewhere in the lifecycle. At the beginning of stage 2 of our project, the whole team was novice in part of the applied technology such as the used for the presentation layer (OpenLaszlo, 2009). Employing new technology in any project implies certain inherent risks, so an adequate technology management is a precondition for a successful software development project (Ould, 1998). Although a training period in OpenLazlo was carried out, it resulted insufficient since a low quality standard of interfaces was obtained at the end of stage 2. After auditing the system (stage 3), it was needed to use the services of an external team experienced on such technology to improve the functionality of the presentation layer. However, this approach also failed, therefore, the team also decided to develop all interfaces left.

The use of CC agile method was decided at the beginning of stage 3 (Section 2), and additional practices from Scrum, XP and UP were included in order to reinforce the safety zone of the project. Special care was taken to train the new team member. It was done by coaching initially, complemented by the *side-by-side programming* practice of CC. Benefits of automatic tests are evident. Manual tests were preferred in the context of a small team with the expert user readily available for checking results (Figure 1) as well as the need of prioritising *early victories* in a time-constrained condition.

Efficient communication is one of the key issues of AM (Abrahamsson et al., 2002; Larman, 2004). Different CC practices were applied simultaneously during stage 3 of the project to facilitate an environment where the members worked collaboratively. Among them, it was applied *side-by-side programming*, intensive use *information radiators* and *story cards*, team co-location including the expert user, and also the modification of the office layout (Figure 2). This context helped to share the 'big picture' of the project state and to build a strong camaraderie and team spirit, which definitely were the key drivers to sustain focus and commitment during stage 3.

The delivery of tested working software in an iterative way brought high visibility of project progress. Direct feedback to measure development amelioration was the sponsor's reaction. By the third iteration, he mentioned "now I feel we are on track, new requirements that we detected by knowing the technological possibilities, are well received and attended soon by committed developers". Considering that 120 days before he was almost aborting the project, this last phrase is not a minor team achievement.

The main conclusions are that undetected characteristics of constantly changing requirements during initial inception phase (stage 2 of the project) caused a wrong selection of a waterfall lifecycle. Hence, increasing difficulties during development produced a significant dissatisfaction from both the sponsor and the team putting the development at failure risk. Application of the CC AM (in combination with selected practices from Scrum, XP and RUP) was highly effective in rescuing this web-based whole-farm project, meeting the sponsor's expectations in less than four months. Finally, four additional iterations were necessary to get an alpha prototype (Machado et al., 2010). At present, new simulator functionalities are in progress.

Main deficiencies reported in at the beginning of this case, were sorted out as:

- Deficient communication. Expert domain (and sponsor) was present only six-hours once week. Telephone and e-mail were also used, but they did not result in an efficient mean. During stage 3, with expert user sharing the working place on a daily basis, communication improved significantly.
- Developers were novices on the domain. Associated to the previous point, this caused some tasks were too partitioned and not adequately integrated to the whole system requirement. The daily presence of the expert user helped notoriously to solve most of these problems (*block gone in one day*), reducing major part of domain bugs at each software release.
- Requirements resulted highly dynamic; therefore, most of the tasks required modifications and re-implementations, causing continuous delays to the development process. Once the sponsor/expert user was trained on CC methodology, it was clear to him that new requirements need to be included to future iteration/s (*do not add to iterations* practice). Finally, he was aware about benefits of the methodology, as goals were increasingly met.
- Deficient documentation of both, the development process and available functionalities. Software documentation was gradually improved, and an intensive use of information radiators helped to track the project progress.
- No working software was available. More than 3,000 men hour had been invested by the end of stage 2 (counting more than 35,000 and other 12,000 code lines in the application layer and presentation layer respectively) but with a high rate of personnel renewing, therefore, most of the planned functionalities were not working properly. After three months (two weeks of *Exploratory 360°* practice and the timeframe of the first iteration) most part of the planned functionalities were working acceptably including a stable structure and the corresponding documentation

The use of CC resulted highly positive since it improved the communication between team members and as a consequence increases the team flexibility and productivity and maintaining focus on those tasks more relevant to the project.

Acknowledgements

To the National Agency of Science and Technology of Argentina, who funded most of this development (PICTO 22926/06) and its continuation (PICT Start Up 0184/07), and to Dr. Alistar Cockburn for reading our early draft and offering valuable advice.

References

Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J. (2002) 'Agile software development methods: review and analysis', *Espoo 2002*, VTT Publications, Vol. 478, p.107, Oulun Yliopisto, Finland.

Bass, L., Clements, P. and Kazman, R. (1998) *Software Architecture in Practice*, Addison-Wesley.

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

Berger, H., Machado, C., Copes, M., Ponssa, E. and Auza, N. (2002) 'Modelo dinámico simple de sistemas de cría (Criasim)', *Revista Argentina de Producción Animal*, Vol. 22, pp.342–345.

Brennan, D. and Gooday, J. (1998) 'Whole farm modeling using ExtendTM simulation software', *Proceedings of the bioeconomics workshop. Australian Bureau of Agricultural and Resource Economics (ABARE)*, pp.65–75.

Cockburn, A. (2004) *Crystal Clear: A Human-Powered Methodology for Small Teams*, Addison-Wesley.

Doyle, C.J., Baars, J.A. and Bywater A.C. (1989) 'A simulation model of bull beef production under rotational grazing in the Waikato Region of New Zealand', *Agricultural Systems*, Vol. 31, pp.247–278.

Ferrari, R.N. and Madhavji, N.H. (2008) 'Architecting-problems rooted in requirements', *Information and Software Technology*, Vol. 50, pp.53–66.

Forbes, T.D.A. (1988) 'Researching the plant-animal interface: the investigation of ingestive behavior in grazing animals', *Journal of Animal Science*, Vol. 66, pp.2369–2379.

Holzworth, D., Meinke, H., DeVoil, P., Wegener, M., Huth, N., Hammer, G., Howden, M., Robertson, M., Carberry, P., Freebairn, D. and Murphy, C. (2006) 'The development of a farming systems model (APSIM) – a disciplined approach', available at http://www.iemss.org/iemss2006/papers/w4/Holzworth.pdf (accessed on 21 April 2009).

Hsqldb (2009) 'Lightweight Java SQL database engine', available at http://hsqldb.org/ (accessed on 21 April).

Illius, A.W. and Hodgson, J. (1996) 'Progress in understanding the ecology and management of grazing systems', in Hodgson, J. and Illius, A.W. (Eds.): *The Ecology and Management of Grazing System*, CAB International, Wallingford.

Johnson, C.W. and Holloway, C.M. (2006) 'Questioning the role of requirements engineering in the causes of safety-critical software failures', *The 1st Institution of Engineering and Technology International Conference on System Safety*, 6–8 June, pp.352–361, London.

Krahl, D. (2002) 'The extend simulation environment', *Proceedings of the 33rd Conference on Winter Simulation*, December, pp.205–213.

Larman, C. (2004) *Agile and Iterative Development: A Manager's Guide*, Addison-Wesley.

Loewer, O.J. (1998) 'Graze: a beef-forage model of selective grazing', in Peart, R.M. and Bruce, R.C. (Eds.): *Agricultural Systems Modeling and Simulation*, pp.301–417, Marcel Dekker, University of Florida.

Machado, C. (2004) 'Field and modeling studies of the effects of herbage allowance and maize grain feeding on animal performance in beef cattle finishing systems', in Unpublished PhD Thesis, directed by Morris, S.T., p.271, Massey University, New Zealand.

Machado, C.F., Morris, S.T., Hodgson, J.M., Arroqui, P. and Mangudo, C. (2010) 'A web-based model for simulating whole-farm beef cattle systems', *Computers and Electronics in Agriculture*, Vol. 74, pp.129–136.

Masticola, S.P. (2007) 'A simple estimate of the cost of software project failures and the breakeven effectiveness of project risk management', *First International Workshop on the Economics of Software and Computation*, 20–26 May, Minneapolis, USA.

McCall, D.G., Sheath, G.W. and Pleasant, A.B. (1994) 'The role of system research in animal science', *Proceeding of the New Zealand Society of Animal Production*, Vol. 54, pp.417–421.

Open Workbench (2009) 'Open-source project scheduling', available at http://www.openworkbench.org/, (accessed on 21 April 2009).

OpenLaszlo (2009) 'An open source platform for the development and delivery of rich internet applications', available at http://www.openlaszlo.org/ (accessed on 21 April).

Oracle XE Express Edition (2009) 'Database based on the Oracle Database 10g Release 2 code base', available at http://www.oracle.com/technology/products/database/xe/index.html (accessed on 21 April).

Ould, M. (1998) *Managing Software Quality and Business Risk*, John Wiley & Sons, San Francisco.

Pannell, D.J. (1999) 'On the estimation of on-farm benefits of agricultural research', *Agricultural Systems*, Vol. 61, pp.123–134.

Pearson, C.J. (1997) *Agronomy of Grassland Systems*, 2nd ed., p.222, Cambridge University Press, New York.

PICTO 22926 (2006) 'Desarrollo y evaluación de un simulador dinámico clima-dependiente de empresas ganaderas predominantemente pastoriles', dir. Machado, C.F., funded by Argentinean National Agency of Science and Technology No. 029/2006, ANPCyT-FONCYT.

Schwaber, K. and Beeddle, M. (2002) *Agile Software Development with Scrum*, Prentice-Hall.

Shaw, M. and Garlan, D. (1996) *Software Architecture, Perspectives on an Emerging Discipline*, Prentice-Hall.

SmartCVS Foundation (2009) 'A graphical SVN client', available at http://www.syntevo.com/ (accessed on 21 April).

Spring Framework (2009) 'Platform to build and run enterprise Java applications', available at http://www.springframework.org/ (accessed on 21 April).

Standish Group's The Chaos Report (2004) 'Extreme chaos', available at http://net.educause.edu/ir/library/pdf/NCP08083B.pdf/ (accessed on 21 April 2009).

Thayer, R.H., Dorfman, M. and Bailin, S.C. (1997) *Software Requirements Engineering*, IEEE Computer Society.