Agile Management: Strategies for Developing a Social Networking Site for Scientists

Yuwei Lin*, Meik Poschen*, Rob Procter*, Alex Voss*, Carole Goble**, Jiten Bhagat**, David De Roure***, Don Cruickshank***, Mark Rouncefield****

*ESRC National Centre for e-Social Science, University of Manchester

**School of Computer Science, University of Manchester

***School of Electronics and Computer Science, University of Southampton

****Computing Department, Lancaster University

Introduction

Research 2.0 (or Science 2.0) is the term commonly used to describe Web 2.0-based platforms for supporting collaboration in scientific research (e.g., Waldrop, 2008). This paper, based on our experience of developing myExperiment[1], a site that enables scientists to share digital resources associated with their research (De Roure, Goble and Stevens, 2007), aims to identify and detail good practice for the development of Research 2.0 sites. We are especially interested in explicating how the project is managed so as, on the one hand, to maintain rich user engagement in the face of uncertain and evolving requirements and to exploit the malleability of Web 2.0 technologies, while, on the other, keeping the project on 'track'. Our interest then is in 'agile management', with how the flexibility and responsiveness encouraged by 'agile' approaches and facilitated by ICT tools can be combined with managerial requirements for coordination, measuring progression, identifying and meeting targets and so on; and with the organizational rationale and consequences of 'perpetual beta'.

myExperiment is funded under the second phase of the UK JISC Virtual Research Environments programme[2] and aims to enable scientists to share digital resources associated with their research, in particular, to share and execute scientific workflows through familiar social networking and social collaboration features. To achieve its aims, the myExperiment project team have adopted a so-called 'agile' approach to software development. For the project team, whose members are distributed across multiple sites (and often mobile), making an agile approach work in practice means relying on a repertoire of mechanisms, deployed at different frequencies (daily, weekly, monthly), involving different combinations of team members (developers, managers, users), focusing on different project aspects and timescales,

[1] www.myexperiment.org

[2] www.jisc.ac.uk/programme_vre.html

using a variety of communication modes and tools, artefacts and structuring devices to achieve the orderly and smooth running of the project.

The records of management meetings, the logs of the developers' daily Skype chat and the mailing lists postings provide a rich project 'biography', documenting over time the origins, evolution and fate of different requirements, the various contingencies that arose and how these were managed as the project unfolded. Drawing on these accounts of project meetings and interactions between team members, we discuss and explicate the significance of activities such as, *inter alia*: reporting, coordinating, scheduling, selecting, prioritising, reviewing, sense-making, problem-solving, revising, refining, refactoring, testing and so on, for the project team's ability to stay on 'track' in the face of constant change, in the context of a distributed team and with the various communication mechanisms and tools to hand.

Agile Approaches

Agile approaches are explicitly aimed at dealing with the problem of uncertain requirements, and make the best use of a short development lifecycle. The Manifesto for Agile Software Development (Beck *et al*., 2001) characterises agile methodologies as "better ways of developing software". Unlike traditional phased design methodologies, agile approaches value interactions over processes and tools, collaboration with users and production of comprehensive documentation. Although there is no universal definition of agile methodologies that can be applied to every domain and every case, some have become more popular than the others over time. Extreme programming (Beck, 2000), for example, focuses on minimisation of risk during the development process (largely through test-driven development) and employs strategies often opposed to traditional systems development and project management wisdom. Among its key elements are a focus on working code, involving early release and short release cycles and an incremental planning approach that allows changes to be made according to evolving circumstances and user requirements. Such agile approaches are particularly useful for developing what is now referred to as a 'perpetual beta', a term for software which is undergoing continuous refinement and is used by developers in order to allow them to constantly release new features that may not be fully tested (Morris, 2006). O'Reilly describes the concept of perpetual beta as follows:[3]

- Services, not packaged software, with cost-effective scalability
- Control over unique, hard-to-recreate data sources that get richer as more people use them
- Trusting users as co-developers
- Harnessing collective intelligence
- Leveraging the long tail through customer self-service
- Software above the level of a single device
- Lightweight user interfaces, development models, and business models.

In this paper we will examine how the myExperiment team's pursuit of a perpetual beta is managed. The team has followed a user-driven, agile approach of scientific software design. This approach consists of six key principles which are discussed in De Roure and Goble (2009): *fit in, don't force change*; *jam today and more jam tomorrow*; *just in time and just enough*; *act local, think global*; *enable users to add value*; and *design for network effects*. Derived over several years of experience of developing e-Science applications and further

3 http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=4

evolved during the course of the myExperiment project, De Roure and Goble argue that embedding users and developers in a project and having them work closely together, starting from addressing local users' needs, and not reinventing technologies are the best way of achieving a system that will be useful for scientific communities.

Written using the Ruby on Rails Web application development framework and released under the BSD license[4], myExperiment, like many other open source software projects, represents a "middle-ground between the conventional, highly formalised software engineering and 'hacking' that inscribes collective improvisation" (Mackenzie and Rouncefield, 2002). The agile approach adopted by the myExperiment team affords a balance between structured orderliness that ensures the project as a whole is delivered on time and improvisation that enables responsiveness to new and changing user requirements, and exploitation of a rapidly expanding array of Web 2.0-based components and technologies. In the following, we will unravel the agile management contextualised in myExperiment by looking at how the team performs agility when dealing with everyday practicalities. We evaluate this against a framework based on the orderly character of project work proposed by Button and Sharrock (1996), namely *phasing*, *orienting to the project as a totality*, *the methodic handling of tasks*, *making sure the documentation gets done* and *measured progressio*n.

The myExperiment Approach

A range of mechanisms and tools have been employed and used interchangeably to help myExperiment team members to interact, orientate to collaborating and to deliver value quickly. Some of these are face-to-face meetings and others are computer-mediated to cope with distributed team work. To coordinate activities day-to-day, the developers working from different sites at Manchester, Southampton and elsewhere have a one-hour Skype chat daily at 5pm to test changes, discuss and review technical strategies, priorities and schedule changes, review new requirements and various options for realising or rejecting them. Weekly meetings are for core members in managerial positions to track development and re-evaluate priorities. Monthly team meetings allow a wider workflow community (e.g., developers from the myGrid project) to take part and meet face-to-face. Since members are located at different places (Manchester, Southampton and elsewhere), collaboration tools such as telephones, instant messaging tools are all used at these meetings. and for ad hoc interactions on a daily basis. A wiki provides a persistent and shared semi-formal record for notes, task lists and decisions. Additionally, team members have frequent meetings with users.

The myExperiment developers, together with other supporting members, meet up every month (often just before the monthly management meeting) for a "hackfest" designed to accelerate development and focus on getting key requirements translated into code. The hackfests last between two and three days and are a great opportunity for the development team to sit in one room and solve problems in a truly collaborative fashion, sometimes bringing in other colleagues to assist. Two mailing lists (one open for general discussion, one limited to the core team) and a wiki (with open and secure area) are also available. The open mailing list has a dual role, serving as a mechanism for users to comment on existing features and raise new requirements, and for developers to announce progress and discuss in an open way with users how new requests will be dealt with. The myExperiment team is embedded within a larger group which is delivering workflow tools, specifically the Taverna workbench

[4] The myExperiment source code is available on RubyForge in a subversion repository. See http://rubyforge.org/projects/myexperiment/

(Gil *et al*., 2007) and the myExperiment project is associated with many existing projects related to the Taverna workflow editor, collectively constituting the 'myGrid family'. Within the myGrid family, a kind of 'inner circle' of users with more specific knowledge is especially valuable for providing feedback on myExperiment. As we illustrate below, it is in these ways and through these means for managing the project while involving users actively in the process, that the team succeeds in keeping the development of myExperiment moving constantly and rapidly forward as an almost 'perpetual beta'.

In order to get a sense of the rhythm of the project and familiarise themselves with its activities, the first three authors were subscribed to myExperiment mailing lists (both the internal and external ones), and conducted participatory observation at the developers' daily 5pm Skype Chat and face-to-face meetings. From the Skype logs, we can easily identify the core participants (usually 3-5 developers including the project manager) and the structure of the meeting which is mostly informal and chaired by the project manager. The topics are based on the developers' daily activities, informing each other of progress, making sense of problems together and solving them. It usually begins with greetings and social chat, and then goes into the main topic. The conversation flow is fast-paced with participants very used to typing and instant-messaging. It is interesting to note that the project team prefers typing over talking when using Skype. One reason may be that Skype, in this case, functions like a shared wall/whiteboard, allowing participants to draw on information from as many sources as possible (URLs, multi-media objects, copy and paste email texts). In the following, we will mainly draw on Skype logs, and observe how an event observed here evolves and subsequently migrates over different communication channels as different participants become involved.

Findings

The agile approach to project management adopted by the myExperiment team is both planful and flexible in order to both adapt to evolving requirements and to deal with problems efficiently as and when they materialise. The planning can be seen in the chosen mechanisms (regular meeting schedules, use of specific forms of communication which members are required to use etc). The flexibility is embodied in negotiations over problem-solving (e.g., the definition of the user community, selection of reusable code). We pay particular attention to the five activities that Button and Sharrock identify as a means to analyse how the myExperiment team conducts their work, and how 'practical solutions are generated through suspension, negotiation, reinterpretation and relaxation of rules and official procedures'.

Ordering the project

In their paper on the organisation of collaborative design and development in software engineering, Button and Sharrock (1996) point to 'the project' as a visible and discoverable set of formatted organisational arrangement within and through which software engineers coordinate their work and make it both personally and organisationally accountable. In viewing 'myExperiment' as a 'project' we also wish to highlight these ordering features, documenting how the myExperiment team achieve the formatted arrangements of the project and how they routinely and regularly display an orientation to these arrangements in the practical way they order and accomplish their work. Of course, such ordering work does not in itself automatically ensure the orderliness of work or provide remedies for all conceivable contingencies, instead the project structure and plan is an achievement, constantly worked at, referred to and amended as myExperiment team members respond to the various contingencies of their work.

Button and Sharrock (1996) document a number of devices for ensuring the orderly character of work, all of which are clearly noticeable in the records of myExperiment team discussions. 'Phasing', for example, ensures that necessary tasks are adequately completed and provides for the various roles and actions associated with the interdependence of activities as well as providing a means for the recognition of incomplete stages. The 'methodic handling of tasks' provides for some kind of system, some kind of order, in the confrontation and elimination of problems as they arise in the project. 'Measured progression' refers to procedures and devices – organisational metrics – for documenting how much of the project has been done and what remains; checking work against schedules, 'what have we done, what do we need to do next, how far have we got' and so on. Finally, 'orienting to the project as a totality' provides a method for project teams to keep each other's progress in view and make it visible, accountable to others, obviously vital in such a distributed project.

According to Button and Sharrock (1996), "the perception of interrelationship between the activities of project participants and the objectives of the project are a problematic feature of project work and that a critical component of the achievement of an orderly relationship between the two is the provision of opportunities for the participants to orient to 'the project as a whole'" (p.378). They rightly point out that the key to a successful project is "not merely a matter of disposing of the individual's work in hand, but of carrying it out as work which is done in orientation to the project's needs, problems and objectives" (p. 378). Within myExperiment, a repertoire of mechanisms and devices have been deployed to routinely and frequently cross-check the work in progress. These include regular scheduled meetings (face-to-face, Skype, telephone), SVN[5] and mailing lists. Since the team members are distributed across multiple sites (not just at Manchester and Southampton but also mobile/remote), most of these activities are made possible by the adoption of a range of communications channels and the participants' competence in moving fluidly between them. Here is an example of a user suggestion originating on the external mailing list. On November 13, 2007, the project manager announced on myexperiment-discuss that "myExperiment has officially moved to open beta. [The developers] did the DNS update yesterday afternoon so www.myexperiment.org is now live." Users responded with feedback an hour later:

Looks good, some early feedback from people here at MIB.
[1] Can / should myExperiment also build a Facebook app* (not a group)? This wouldn't duplicate myExperiment functionality but would provide some subset of it.
[2] Is there a myExperiment API (I remember this came up before with the Google widget stuff)
[3] Is there API documentation anywhere?
I think these questions might have been asked before, but I wasn't around to hear the answers...

Tracking the chronology of this requirement request shows it featuring in face-to-face management meetings, focus group meetings between users and developers, developers' daily Skype chat, developers' hackfest, and other brainstorming and information sharing on mailing lists, and illustrates how the project team swiftly and smoothly use these different channels to communicate, update and prioritise their work. The API was published and ultimately a 'FaceBook app' for myExperiment was contributed to the public discussion list by a myExperiment user. The following example concerns updating the front page (also called 'homepage'). A first version with a screenshot has been distributed on the internal mailing list and this is picked up in the daily developers chat on Skype:[6]

[17:19:54] PM: let's return to that when we've discussed some other things
[17:20:05] DevA: ok

[5] A an open source software version control system and repository.

[6] The Skype logs have been anonymised: PM= Project Manager; Dev= Developer.

[17:20:05] PM: because i think we need to do so prioritisiation
[17:20:08] DevA: yeap
[17:20:33] PM: ok, home page now, then performance
[17:20:44] PM: you'll have seen my mail to top secret [an internal mailing list]
[17:21:08] PM: saying these two things matter esp for the people who visit us after seeing nature, ieee, thes, or when we do our press release
[17:21:25] PM: I'm worried we tend to focus more on existing users than new sometimes
[17:21:36] DevA: well, we did do that big users/emails work
[17:21:36] PM: so this is a bit of focus on the new!
[17:21:40] DevA: which applies to new users
[17:21:45] PM: yup
[17:21:49] DevA: but yes, front page need redoing
[17:21:59] PM: so i think the front page is a big improvement
[17:22:02] PM: and good to have it up
[17:22:12] DevA: I've spent about 90 minutes with users just before this
[17:22:18] DevA: and based on their feedback i have a new version
[17:22:21] PM: I'd really like to know what users would want to see there to make it inviting
[17:22:23] DevA: i've emailed to topsecret
[17:22:28] PM: excellent

In our preliminary analysis, we have identified three other events (detailed below) that also demonstrate the careful balancing act required between planning and agility, between control and flexibility, in order to keep the direction of the project open and to capitalise on new ideas (for example, a continuous stream of users' suggestions for enhancements) while, at the same time, preserving a sufficient degree of oversight and control on the part of the project management.

Technical strategy

The development of workflow 'enactment' was ongoing when we started collecting the Skype chat log. Thus, we were able to have a diachronic view of the issue and the entangled socio-technical issues involved. The development of enactment highlights some fundamental questions about the project, including the technologies to be used (Grid or Web?) and the nature of the user community being targeted (Is there only one universal myExperiment user community or many fragmented ones?). Such questions were open issues at the start of the project and would only be answered as the project moved forward. As these technical issues are raised and resolved so project members are also required to orient to the project as a totality.

Enactment is a feature request from some Taverna users to be able to start/execute a workflow on myExperiment directly. To be able to do so, however, several questions need to be addressed by the project team: is the enactment code going to be useful for the wider open source software community? Is this function going to be useful for people using workflow editors other than Taverna (e.g., Kepler)?

Below is an excerpt from the Skype chat log on 12 February. The chat, taking place when the project manager was travelling, was brief but significant. It illustrates the flow of information between different communication channels (emails and Skype chat) and how these are brought to bear on issues, such as which technologies (e.g., applications, APIs) to use, whether to reinvent or re-use, and how time constraints are factored into design decisions:

[17 : 34 : 50] PM: i have 15mins.
[17 : 35 : 03] PM: is there anything in particular you need me for in this window ? :).
[17 : 35 : 14] PM: (I mean 15min window !).
[17 : 35 : 35] DevA: not really ... quick update from me is that i've played with the prototype enactment code .
[17 : 35 : 44] PM: well done !.

[17 : 35 : 46] PM: saw the emails.
[17 : 35 : 50] DevA: and now going to start work on the experiment/enactor stuff.
[17 : 35 : 55] PM: you're doing the right thing.
[17 : 36 : 01] PM: gr8.
[17 : 36 : 02] DevA: ok great.
[17 : 36 : 21] PM: I'll shout when i think there is somethign useful in the grid job world that we should use rather than reinvent.
[17 : 36 : 25] PM: we're ok at the mo.
[17 : 36 : 32] DevA : ok that would be great.
[17 : 36 : 32] DevA: thanks.
[17 : 36 : 47] PM: [anonymous] is closer to grid workflows.
[17 : 36 : 54] PM: but i know all the job stuff pretty well.
[17 : 36 : 57] DevA: i purposely tried to make the jobs design simple enough to build in the timeframe now.
[17 : 37 : 01] DevA: ok.
[17 : 37 : 02] PM: exactly right.

User requirements and shifting community

In dealing with the questions raised above, we observe a constant negotiation, re-defining and re-discovering of user requirements. The project team addresses the question of whether the features developed are solely for Taverna workflow editor users or for a wider workflow community. The team try to anticipate how to make the features more generic for a wider group of users, and this is supported by working with 'user advocates' in the target communities and other colleagues in the scientific workflow community. As one can see from the dialogue below (from the Skype chat on 15 Feb.), having pilot/power users test and comment on developments was crucial. These pilot users are provided by the myGrid family. A lot of time the development work ties in with Taverna training, MSc courses and other demo events in which these pilot users are involved. However, these pilot users do not necessarily represent the whole future (emerging) user community. The team has to continue to anticipate their potential users (including people using different workflow editors, and from different disciplines and backgrounds), re-define the boundaries of their user community (although the local pilot users are good reference points) and recruit new users (e.g., at conferences). To do so, they (loosely) categorise their potential users and build different scenarios on these imaginings. However, the boundary of the myExperiment user community would be redrawn only if other requirements can be successfully negotiated, subject to the constraint that existing Taverna users' needs are not compromised too much.

[17 : 15 : 16] DevA: the last i did on the taverna input editor was show it to [anonymous] and the scufl extension i did.
[17 : 15 : 25] DevB: i think with the runner/runnable model we might even be able to support another workflow engine soon.
[17 : 15 : 28] DevA: he wasn't happy with the modifications to scufl , so the editor ended up being generic still.
[17 : 15 : 55] DevA: i suspect that as we get the input definition into whereever it fits into , we can lose most of those `` - > list " buttons etc.
[17 : 16 : 01] DevB: yeap , i think the scufl extensions instead will become annotations on input ports using the new annotation model.
[17 : 16 : 08] DevA: which would make it much easier on users using the input editor.
[17 : 16 : 11] DevA: yeah.
[17 : 16 : 19] DevA: that's the problem [anonymous] had with it.
[17 : 16 : 24] DevA: but now is the right time to do that part.
[17 : 16 : 38] DevB: Z – I'm not sure ... because a basic principle is that there is no limit on the amount of lists of lists yuo can have for one input port.
[17 : 16 : 48] DevA: i know.
[17 : 17 : 11] DevA: but being too generic leads us into the problem where casual users have to create the structure , not just the actual inputs.
[17 : 17 : 28] DevB: to do which part ? the annotation model stuff ?.

[17 : 17 : 29] DevA: i was under the impression that we were going to give people canned example input sets.
[17 : 17 : 50] DevA: being able to say “input a” is just a string and “input b” is a list of strings.
[17 : 18 : 00] DevB: we could do ... a discussion arose today about storing example input data sets.
[17 : 18 : 13] DevA: that discussion has been had a few times over the project.
[17 : 18 : 18] DevB: yeap.
[17 : 18 : 21] DevB: i expect so.
[17 : 18 : 33] DevB: what we’ll do is have the users play with what we have and then see what they ask for.
[17 : 18 : 35] DevA: anyway , i think we most of the pieces to fit together.
[17 : 18 : 43] DevB: yes.
[17 : 19 : 10] DevB: ok , so just to point out ... one aspect of all this enactment code is error handling.
[17 : 19 : 20] DevA: ok.
[17 : 19 : 31] DevB: thats obv a big part of the user experience.
[17 : 19 : 37] DevA: yeah.

As a member of the broader myGrid family, the organisation of myExperiment is influenced by interdependences between various projects. There are also issues about reusing code and producing reusable code that means the team has to compromise what they can accomplish since this is dependent on other available technologies and other possible requests.

Solving an early service issue

myExperiment is engaged in outreach to new users and domains and is committed to keeping a service in use which is also under continuous development. In this undertaking it is inevitable (and to some extend intended) that technical issues will emerge that have to be dealt with. With the numbers of users growing quickly with the start of 2008, the old myExperiment server experienced problems in dealing with the load imposed upon it. This coincided with a new use pattern of concurrent access, as in a Taverna workshop where a group of users did the same thing simultaneously, something for which the codebase at that time was not optimised. Because of its impact on the service, the matter was prioritised and the existing server subsequently replaced by more powerful hardware and the codebase was optimised using standard web scalability techniques in a progressive manner (in keeping with the agile nature of the project) in order to meet the environment requirements. This was done with the view that it would be an ongoing piece of work throughout the lifetime of the project. In the following, we show how the myExperiment team responded to and devised a plan to successfully address and solve the service issue.

The service issue occurred on two successive days, with the service slowing down. On the first day, it was reported as it happened via email on the internal mailing list, with an immediate reaction of a team member being to restart the software, and the nature and impact of the problem was discussed on the internal email list. On the following day, the service was slow during a Taverna/myExperiment training event (due to the concurrent access), which also was reported promptly by email on the internal mailing list and through an additional personal report to the team by the person in charge of the event. There had been no 5pm Skype chats that week because of other events and staff on leave. Therefore, the Skype chat taking place on the second day of the incident was the first occasion to discuss this problem. The PM prepared an agenda in order to tackle this efficiently:

1. Post mortem for both failures
2. Human process for responding to server failure
3. Automated process for responding to server failure
4. Graceful failure modes
5. Service monitoring
6. Testing plans
7. Actions

Following the developers' established practice, the Skype chat served as a mechanism for combining information exchange and discussion together in the lead up to collective analysis and decision-making. Because of the importance of the problem, the PM ended the chat with a summary (see excerpt below), and circulated a detailed description of outcomes, actions and a management level summary via the internal mailing list after the chat.

[18:05:00] PM: just going to summarise against agenda:
[18:05:23] PM: 1. post mortem. we looked at the bodies in the crime scenes, then we presented some theories to inform the investigation
[18:05:46] PM: 2. human process. yes, we'll stay human in the loop for now, but will look at a cronjob for the 3am solution
[18:06:05] PM: 3. we'll implement health check-in scripts and monitoring by human
[18:06:18] PM: 4. not discussed. probably can improve error reporting
[18:06:31] PM: 5. DevC will work on jmeter
[18:06:41] PM: 6. not discussed, one for next week.
[18:06:45] DevA: I'll process the production log a bit more
[18:07:03] PM: who is going to do health check scripts?
[18:07:10] DevA: i'l do those
[18:07:14] PM: thanks DevA
[18:07:20] PM: ok, we're sorted for now
[18:07:37] DevB: we need to also schedule in the continual performance work - adding caching to other parts of the site (eg groups page) and the eager loading optimisations on models
[18:07:40] PM: Be ready for a roomful of people hitting it just after 9pm
[18:07:47] DevA: ok
[18:08:04] PM: DevB - yes, and we need to schedule a daily review
[18:08:11] DevB: ok

In this example, the Skype chat shows a mixture of formally structured elements (such as the agenda and the summary containing detailed outcomes and actions widely circulated widely around the project) and the creative hacking and talk (e.g., the, at times, colloquial language, inside talk and used metaphors, like 'crime scenes' which would lead to a more lively discussion of issues and approach to solutions in a relative short time). High level management decisions are not rendered in the chats; instead, they are endorsed by agile solutions that emerge from the chat that provides a communication space for managing the structured plan and improvisation, while allowing ideas to be aired and discussed. This was especially evident in this case in the thoughtful discussion of the important and time-critical early service problems that were threatening the project and, accordingly, given absolute priority. Here the inherently flexible week-to-week development decisions meet with the rigid, more formal project management plans in a very explicit way.

Conclusions

Our initial interest was primarily in understanding the character of myExperiment as a 'project' – how it actually 'gets done' and how the obvious, visible orderliness of the project is practically achieved. Viewing myExperiment as a practical, ongoing achievement, and concentrating on the everyday, mundane aspects of keeping a project going, we place a particular emphasis on various kinds of 'ordering work' that occurs at a number of levels and draw attention to the various forms of communication – through email, Skype chat, etc.. Given the dispersed or 'virtual' nature of the myExperiment team, the project necessarily attaches considerable importance to these issues of coordination, communication, accountability and awareness. Although there is not enough room to develop this analysis here, another related area of interest, given the emphasis on social networking, concerns the evolving nature of the 'community' involved in myExperiment, with some interesting clues as to exactly how a scientific community is built and sustained. In particular, what we see at

work in this early analysis of the emails and Skype logs, etc. is how the defining features of community, the key features of boundaries, relationships and change, are identified, negotiated and renegotiated in public throughout the course of the project.

In this paper, we have examined the different ways in which the myExperiment team orientate themselves to project work. Instant messaging tools allow the team members located at different places to have frequent interaction; complex and interdependent development work can be coordinated, progress can be seen achieved, and problems can be tackled efficiently. What we see is how these different kinds of interaction come into play to deliver the characteristic of a 'perpetual beta', i.e., to deliver value to users quickly.

Of course, we have yet to see how the myExperiment project is coordinated and orchestrated when more contributions are made by outsiders, and how much their methods resemble those used by other open source software projects; nor do we yet understand how such an agile approach can be used in a much bigger project. Nevertheless, our studies of the use of various communication and structuring devices suggests that agility and top-down management approaches are not necessarily contradictory or conflicting, but well-balanced. Given the principle that software engineering has to deliver and eliminate as many problems and risks as possible, the agile approach that myExperiment team adopts seems well suited to facilitate situated, accountable, agile management.

References

Button, G. and Sharrock, W. (1996). Project Work: The Organisation of Collaborative Design and Development in Software Engineering. Computer Supported Cooperative Work: The Journal of Collaborative Computing 5: 369-386

Beck, K. (2000). Extreme Programming Explained: Embracing Change. Addison Wesley.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. (2001). Manifesto for Agile Software Development. URL: http://agilemanifesto.org/ (Retrieved on 23 May 2008).

De Roure D., Goble, C. and Stevens R. (2007). Designing the myExperiment Virtual Research Environment for the Social Sharing of Workflows. In Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, Bangalore, India, 10-13 December, pp. 603-610.

De Roure, D. and Goble, C. (2009). Six Principles of Software Design to Empower Scientists. IEEE Software.

Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L. and Myers, J. (2007). Examining the Challenges of Scientific Workflows. IEEE Computer 40(12): 24-32.

Mackenzie, A. and Rouncefield, M. (2002). How 'hacking' hides a project: from software engineering to open source and back again. Appendix C.

Morris, J. (2006). Software product management and the endless beta. URL: http://jimmorris.blogspot.com/2006_08_01_jimmorris_archive.html (Retrieved on 09 April 2008)

Waldrop, M. M. (2008). Science 2.0: Great New Tool, or Great Risk? Scientific American. URL: http://www.sciam.com/article.cfm?id=science-2-point-0-great-new-tool-or-great-risk (Retrieved on 13 Feb. 2008).