

# Six Principles of Software Design to Empower Scientists

David De Roure, *University of Southampton*

Carole Goble, *The University of Manchester*

**Scientific research is increasingly digital. Some activities, like data analysis, search and simulation, can be accelerated by enabling scientists to write workflows and scripts that automate routine activities. These capture pieces of scientific method that can be shared with others. The Taverna Workbench, a widely deployed scientific workflow management system, together with the myExperiment social web site for the sharing of scientific experiments, have been devised according to six principles of designing software for adoption by scientists and six principles of user engagement.**

Science is becoming increasingly digital, and the scientist's tools are not just the experimental apparatus of the laboratory but also the software apparatus which they use to conduct their research – to analyse data, to search databases, to run simulations and to record their scientific process. New experimental methods – from DNA microarrays to sensor networks in the environment – are generating volumes of data that, without software assistance, just wouldn't get processed.

Watch a researcher at work and we see lots of activity at the computer, using applications, services and data that might be local to the laboratory, the enterprise, or out on the Web. These new research tools and methods are evident across a very broad spectrum of disciplines. Some, like bioinformaticians working with protein sequences, might conduct some of their research entirely *in silico*. Meanwhile, chemists in the laboratory are looking up procedures, designing their experiments, conducting and recording their work, and running simulations, searches and analyses. Climate change researchers are capturing the latest data from radar images and sensor networks, and running and testing their models. Musicologists are extracting features from recorded music as part of musical analysis. Archaeologists are exploring 3D visualisations of their digs.

Some of the processing is highly repetitive – repeating processes with new data, varying parameters, and making small changes to the process. It clearly helps speed things up and makes them more repeatable if some of this scientific activity at the computer can be automated. This in turn lets scientists concentrate on the science rather than the repetitive handling of data through multiple tools and applications. So, increasingly, we see these scientific activities at the computer being 'programmed'. Some disciplines have created scripting solutions, perhaps adopting their favourite scripting languages and creating libraries to handle their discipline-specific data, while others adopt workflow systems which automate the processing of data through a series of processing stages.

Taverna is one of many scientific workflow management systems (see *Workflow Management Systems* on page 2) and supports what we might describe as 'application-level workflows', as opposed to some other systems which focus on scheduling tasks across computing resources [1]. Emerging from the UK's e-Science programme, Taverna is used extensively by scientists across a range of Life Science problems: gene and protein annotation; proteomics, phylogeny and phenotypical studies; analysis of microarray data and medical images; high throughput screening of chemical compounds and clinical statistical analysis. Increasingly it is being adopted in other disciplines too.

Workflows are leading to new scientific outcomes. For example, Paul Fisher and colleagues have reported on the benefits of being able to “systematically analyse any results we obtain without the need to prematurely filter the data for human convenience” [2] – which enabled them to identify a candidate gene that had previously been missed. Workflows are systematic and unbiased, and they capture data analysis methodologies explicitly. The popularity of Taverna and other workflow environments is changing not only how scientists program their experiments, using high level operators in terms of the problem and steps in terms of the scientific operations, but also how their experimental methods are propagated and communicated through, and between, communities.

## Scientific Workflow Management Systems

For an overview see : *Workflows for e-Science: Scientific Workflows for Grids* Taylor, I.J.; Deelman, E.; Gannon, D.B.; Shields, M. (Eds.) 2007

**Taverna** Taverna is an open-source workflow tool which provides a workflow language and graphical interface to facilitate the easy building, running and editing of workflows over distributed computer resources. The Taverna Workbench allows users to construct complex analysis workflows from components located on both remote and local machines, run these workflows on their own data and visualise the results. [taverna.sourceforge.net](http://taverna.sourceforge.net) and [www.mygrid.org.uk](http://www.mygrid.org.uk)

**Triana** An open-source problem solving environment developed at Cardiff University that combines an intuitive visual interface with powerful data analysis tools. Already used by scientists for a range of tasks, such as signal, text and image processing, Triana includes a large library of pre-written analysis tools and the ability for users to easily integrate their own tools. [www.trianacode.org](http://www.trianacode.org)

**Kepler** The Kepler project's overall goal is to produce an open-source scientific workflow system that allows scientists to design scientific workflows and execute them efficiently using emerging Grid-based approaches to distributed computation. Kepler is based on the Ptolemy II system for heterogeneous, concurrent modelling and design. [kepler-project.org](http://kepler-project.org)

**Pegasus** Pegasus (Planning for Execution in Grids) is a workflow mapping engine developed and used as part of several NSF projects. Pegasus bridges the scientific domain and the execution environment by automatically mapping the high-level workflow descriptions onto distributed infrastructures such as the TeraGrid, the Open Science Grid, and others. [pegasus.isi.edu](http://pegasus.isi.edu)

### Social Experiments

These workflows and scripts are particularly valuable because they are descriptions of pieces of scientific process. Providing a workflow along with the results of an experiment enables the provenance of the results to be understood and makes those results more reusable. Publishing a workflow enables work to be reproduced or at least correctly interpreted. Along with this value comes a cost – writing scripts and workflows is not always easy. So it’s handy to share scripts and workflows with colleagues so that they don’t have to create them from scratch. But actually the sharing of scientific method through scripts and workflows promises to be more powerful than that, because it means a scientist in one domain can borrow a method from another. There is real evidence of this happening – see *Recycle, Reuse, Repurpose* on page 4.

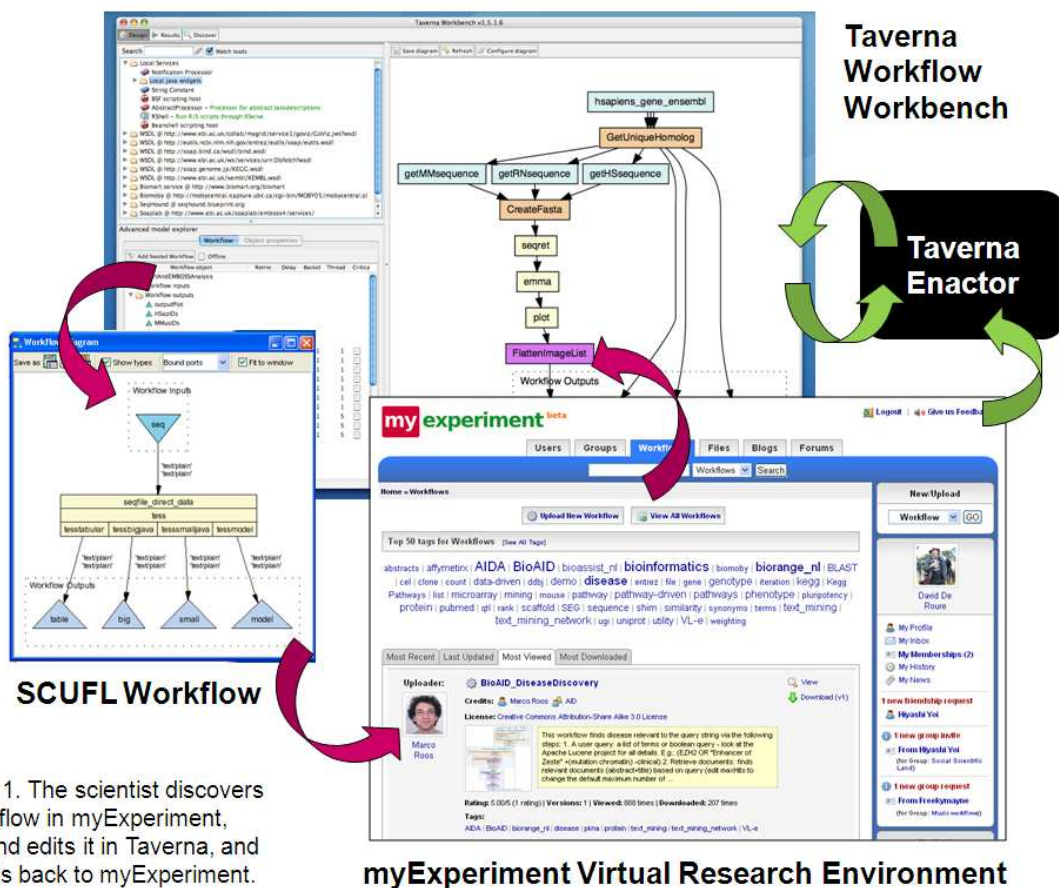


Figure 1. The scientist discovers a workflow in myExperiment, runs and edits it in Taverna, and uploads back to myExperiment.

All these benefits of sharing workflows and scripts motivated our creation of the myExperiment social web site. Using myExperiment, researchers can share these ‘experiments’ whatever their workflow system – Figure 1 depicts the cycle of workflow discovery in myExperiment and editing in Taverna (for example). Although sometimes described as ‘Facebook for Scientists’, myExperiment is different to sites for sharing pictures, slides etc, because it focuses on the special requirements of scientists – such as the need to describe the attribution of work, control visibility and sharing in groups, handle licensing, and work with distributed collections of data. myExperiment couples decoupled communities and brings community intelligence to workflow systems.

Take workflow reliability for instance: the success of a workflow depends on all the services being available and working as expected, which is a difficult challenge. Many of the services linked together as steps in workflows are third party, autonomous, and changeable – so workflows tend to decay. Workflows can call many different services, potentially invoked hundreds of times. By sharing workflows within a community we can achieve community recommendations and maintenance of the workflows, and simply by keeping track of their use the Web site itself can provide recommendations of workflows and services. It also enables us to exchange scripts and workflows for multiple tools, broadening the scientist’s toolkit.

### The Software – Superclients and the Web

Taverna is distinctive for being a ‘superclient’ which anyone can download and run on their desktop or laptop PC, without needing anything extra installed on servers nor any system administration. Any services you can get at from your PC, whether they’re in the enterprise or in the Web, can be plumbed

together by running a Taverna workflow. The idea is that it should be like downloading a Web Browser, except it runs workflows instead of displaying Web pages. There are three main parts to Taverna:

- **The Taverna Enactor.** This is the engine that takes a Taverna workflow and executes it using the data provided by the user, over the services described within the workflow. In its early form the enactor did simple data staging from service to service; it has evolved to support streaming, runtime determination of services from service groups, and numerous extension points for developers.
- **The Taverna Workbench.** This provides a graphical editor for workflows but also, significantly, the means for users to choose services – and it is the availability of services for a particular domain that makes Taverna easy to use in that domain. It is easy to add new services. The workbench provides additional tools; for example, it captures the execution logs of services in order to record the provenance of the results.
- **The Taverna Language.** This is a simple dataflow language called SCUFL (*Simple Conceptual Unified Language*) with implicit iteration constructs, manifest graphically in the Taverna editor and encoded in XML. It is actually a declarative language with its semantic roots in functional programming. It is quite different to languages such as WS-BPEL that support a control flow paradigm.

Meanwhile myExperiment is a quite different piece of software – it's a web based application built on the Ruby on Rails (RoR) platform, following the mores of modern Web 2.0 look and feel, social features and lean APIs. It's not just a single site, like Facebook, YouTube etc; it's also a software package that can be installed independently and separately in a laboratory, and supports the exchange of content between other Web applications and different installations of myExperiment. It's also designed to be extensible, so that it deals not in workflows or scripts per se but in scientific objects. This allows sharing of documents, presentations, service descriptions, notes, ontologies, plans and so forth. More generally, myExperiment deals in experiments, and can be used to glue together heterogenous collections like distributed experimental data or, for example, packs of workflows – these collections are what we call *Encapsulated myExperiment Objects*.

## Software for Science

Taverna has achieved extensive adoption, topping 46,000 downloads from Sourceforge to date and averaging in excess of 40 a day since 2006. myExperiment went from conception to open beta in nine months, with the codebase delivered by a core team of two developers. Now at twelve months it has over 600 users, despite the limited advertising of quite a specialist tool.

So we can proudly say that this software has been embraced by scientists. This is not always the case – we have developed systems in the past that were fine examples of well-designed software but disdained or neglected by their intended users. Scientists have challenging and changeable applications that they really understand but can be hard to communicate or stabilise. They are concerned with getting scientific results rapidly in order to compete with their peers and develop their reputation.

Scientists do want reliability and confidence in their software but, paradoxically, are less concerned with generic solutions that can be replicated amongst their community and evolved systematically.

Software engineers, on the other hand, are motivated by these very things. They want to build the best solution and get that adopted by many other users, but are not the specialists that will use the software. Sometimes we have computer scientists in the mix too, and they also need to compete with their academic peers – by exploring interesting technologies and innovative methods which may well be at odds with the needs of the scientists and software engineers.

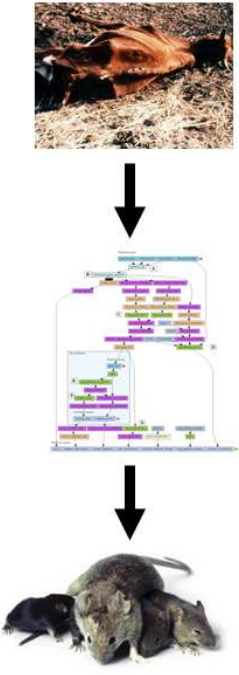
## Our Six Principles of Design for Adoption

Scientists, to be successful, must be fundamentally selfish – Mike Ashburner, a Cambridge geneticist, once said that “Scientists would rather share their toothbrush than their data”. Adoption is based on what the software will do for them. To gain adoption means recognising this. What counts the most? It’s not clever software or innovative technologies but rather content and relevance. During the development of Taverna we established a set of design principles and a working practice with our scientists, further evolved by myExperiment [3]:

**1. Fit in, don’t Force Change.** We made no obligation on service providers to change their services to fit into Taverna. So we rapidly assembled content – more than 3500 service operations covering all the major service providers in the community (there are over 900 databases routinely used in Bioinformatics [4]). Our early users were prepared to tolerate less-than-perfect interfaces because *their favourite service* was available. The user does not have to make any changes in order to use the system – they simply download Taverna and use it, they can use workflows immediately, and they view their data with the same applications as before. Its functionality sits comfortably with their existing environment and practice, and barriers to adoption are kept low at the expense of, for

### Recycle, Reuse, Repurpose Workflows and Know-how

- Paul writes workflows for identifying biological pathways implicated in resistance to Trypanosomiasis (caused by the parasites *Trypanosoma congolense* and *Trypanosoma vivax*) in cattle (see [2])
- Paul meets Jo. Jo is investigating *Trichuris muris*, another parasite but this time in mouse. She is using the same kind of applications and services, and some different data sets
- Jo reuses one of Paul’s workflows **without change**.
- Analysis of the resulting data by Jo identifies some biological pathways involved in sex dependence in the mouse model, believed to be involved in the ability of mice to expel the parasite. It takes two days for the analysis.
- Previously a manual **two year study** by Jo had failed to do this.



*With thanks to Paul Fisher and Joanne Pennock,  
The University of Manchester, UK.*

example, elegant type systems.

In myExperiment our motto is *bring myExperiment to the user* rather than forcing the user to come to myExperiment. So for scientists who are already using web sites and Wikis in their work, we make it easy to bring myExperiment functionality through their existing interfaces. For those who don't, the Web interface is made as familiar as possible and adapted to their needs.

**2. Jam Today and more Jam Tomorrow.** This is about incentives. Incremental development and incremental content give an immediate return for the time and effort invested by a scientist. The *activation energy* required by users to adopt a feature must be matched by the reward gained. It's important to get some core capability and quick wins out promptly. We built our community as we built the software: our scientists and developers travelled on the journey together. Telling our scientists to commit time and effort to something that they wouldn't get any benefit from along the way was pointless; by the time we would have got there they would have another problem or would have found an alternative solution. Once service providers see that their resources would get greater use if they made them more amenable to Taverna, then they will put in the effort.

Taverna has immediate benefit through automation of repetitive and time consuming tasks, and a promise of greater value in the future as more workflows become available and as the user, if they wish, become creators of workflows. myExperiment gives an immediate return by providing examples of workflow others have written and the promise of more workflows and community help.

**3. Just in Time and Just Enough.** This is about delivery. Coupled with the previous point, we learnt not to try to be too smart (always a temptation for computer scientists!) *Be better not perfect.* This enables us to respond to users' immediate needs rather than going away and building a complete solution – which would be more resource-intensive and, in our experience, quite possibly wrong. It is better to solve the problems users already know they have, than to make them wait for solutions to problems they don't yet know they have and perhaps never will.

For example, in Taverna we worked on fancy knowledge-based descriptive techniques for services so that workflows would be composed automatically. It turned out that this wasn't what the users wanted at all. They wanted quick ways of finding a relevant service and then they wanted help for them to build workflows themselves. Building software to power a web site is different to building workflow system software – for one thing, we found it easier to be incremental, to involve a distributed and disparate community of users in the design process, to capture their input and be agile in response.

**4. Act Local, think Global.** Instead of setting out to build a universal workflow system or a universal social networking site, we targeted a community we know really well. Moreover, we picked a few close 'friends and family' – local pioneers who are stereotypical examples of a class of scientists with a class of problems – and just built it for them. We worked really closely with them, and a few key service providers. It turns out that if they are happy, then so are the scientists who are strangers to us. So, Taverna is delivered with sets of services which meet the specific needs of target user communities, and is easy to extend to others. Meeting local needs encourages adoption; making it extensible broadens that. Our experience suggest that *customisation outvotes genericity, extensibility outvotes comprehensivity, and scruffy and flexible outvotes smart but rigid.*

In myExperiment, we took as a principle never to make assumptions about what users want. That we were limited to a core team of two developers made it essential to impose a very simple mantra: *don't do anything unless a user has asked for it.* Our failures occurred when we tried to second-guess a



solution for which there was no clear user problem statement. As Don Wells said, “Don’t Solve a Problem Before You Get to It” [5].

**5. Enable Users to Add Value.** This is about empowerment. Extensibility and customisation are crucial to adoption. The users have the best understanding of the visualisation tools they want to use or the services they want to incorporate. Many users are capable developers, or have developers working with them. Taverna enables users and developers to enhance the system through creating content, integrating services and developing software. It is extensible through new services and through plugins, and these extensions can be shared so that the value accumulates. The Taverna team does not need to do the extensions, but rather it provides the support and training for others to do so.

The small team associated with myExperiment means that *maximal reuse and reusability* is a necessity. We aimed to make as much use as possible of third party code and services, and to make our services available through a simple (RESTful) API published as early as possible, since this enables others to start development and to build functionality mashups. Within hours of releasing the API we had more people developing outside the team than within it. By providing a network of cooperating data services with simple interfaces which make it easy to work with content, we are both providing and reusing services – we support lightweight programming models for ease of integration in loosely coupled systems.

**6. Design for Network Effects.** This is about community. Our users are not just the heroic e-Scientists harnessing computing resources to tackle major scientific breakthroughs, but also the large number of scientists conducting the routine processes of science on a daily basis, the service providers who supply the resources they use, and the community of workflow developers willing to share their know-how. Harnessing this long tail builds a network effect of adoption.

For myExperiment, it must be easy to find workflows and also useful and straightforward to share workflows and add workflows and other scientific assets to the pool. The usage of workflows and services implicitly enables recommendations, as well as the more explicit mechanisms of reviewing and “favouriting”.

These principles are interconnected, of course, and are predicated on well-designed, loosely coupled modular software. Principles 2 and 3 led to our adoption of a perpetual beta software development methodology. Taverna uses a plugin and code manager called Raven which manages extensions supplied by third parties and provides some level of automatic updates. The myExperiment codebase is developed and tested continuously, with weekly updates to the live service. Both projects are just as much – or perhaps even more – about content as software. We were reminded of this when we went open beta with myExperiment and the first bug report that came in was about a spelling error in a user-provided workflow description. Users come to the site for workflows, and – as with Taverna – we have been careful to bootstrap the system so the users have ‘jam today’.

## **Building Trust between Users and Developers**

All these principles depend upon a good relationship between users and developers. It is clear from these principles that Taverna and myExperiment are very user-focused. Taverna started out as an e-Science pilot project in 2001 and one third of the team that built the first prototypes were bioinformaticians. Now the software is delivered and supported by a software engineering team (of

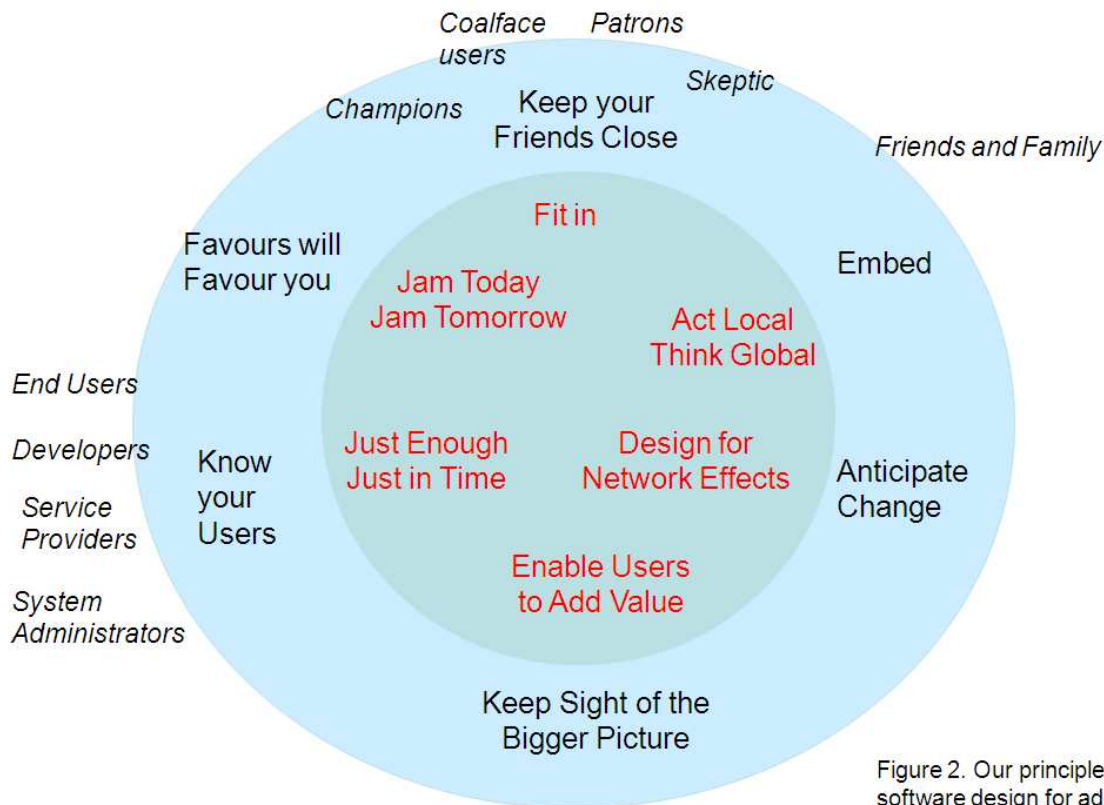


Figure 2. Our principles of software design for adoption (inner circle) and user engagement (outer circle).

six) and a user outreach team (of three bioinformaticians), part of the Open Middleware Infrastructure Institute UK (OMII-UK) and in close collaboration with its user communities. Throughout myExperiment we have operated with two core developers and a larger team around them providing occasional support, including part-time liaison staff for specific research communities.

We have achieved this through our six principles of user engagement, which are also show in relation to our design principles for adoption in figure 2.

**1. Keep your Friends Close.** In addition to community development with local pioneers and external early adopters, OMII-UK operates a scheme of beta testers and ‘Product/Area Liaisons’ (PALs), who are not directly working for OMII-UK but are the eyes and ears in the communities they work in. Taverna has three very active PALs, who are critical to identifying user needs and to developing the community. By choosing these individuals we recognise their worth both to OMII-UK and to their community. In myExperiment, user advocacy was built into the management plan. User workshops (dubbed ‘parties’) are combined with *community champions*. Our ‘friends and family’ users (including sceptics!) and the broader communities are involved at all times – through everyday use of the Web site, through consultations with mockup page designs, and through facilitated user evaluation sessions.

Our advocates are all the day-to-day users of our software – mostly postdoc scientists and postgraduates. In academia, the way to spread software take-up is through these groups of users, not by targeting the great and the good. However, to get their (and community) support they need to be sheltered by sympathetic patrons – their professors and laboratory managers. A more controversial



viewpoint is to pick good scientists who are on the second rung, not the very best. Those who want to overtake their peers will look to new tools to help them and will be more willing to try them out.

**2. Embed.** We embed developers with users and users with developers, getting them to sit side by side for long periods. No amount of requirements documentation or design meetings can really compensate for day to day contact. We started myExperiment by embedding the developers in an end-user laboratory, for first-hand experience of the work environment particularly with respect to sharing and communication, and understanding non-functional requirements.

**3. Keep Sight of the Bigger Picture.** A tendency of developers, and computer researchers, is to get hung up on some point that a user doesn't worry about in the overall scheme of things. Our software is not the only software our users use, and they are only using it to do the science they really want to do. They daily use complex tools they are familiar and comfortable with using metaphors that seem peculiar to someone outside their discipline but are standard within it. Mimic these tools and link with them. The software tools are part of the overall process – we can accelerate science not just by accelerating experiments but also by reducing the time to get to an experiment.

**4. Favours will be in your Favour.** Building trust is a two-way activity, requiring *compromise and favours*. In the early days of Taverna we often ended up writing specific code to help our pioneering scientists get their scientific results. For every pioneer, we 'sacrificed' one developer in support. This could have been argued as wasted effort: of course it was not. The developers learnt about the problem and the users felt we were on their side. The developers got 'jam tomorrow' too, with the patient and added attention of the scientists when it came to evaluating prototypes and designs. By enabling users to add value they also gain ownership of the software; another plus for adoption. Today we still have three outreach workers and we still develop, or help partners develop, bespoke code that leads to a general release (Principle 4). In myExperiment we prioritise tasks and allocate development effort based on the return in adoption.

**5. Know your Users.** Rarely is there one kind of user. For example, for our software we have bioinformaticians and biologists (two distinct user groups); domain expert developers producing applications, service developers, service providers, and system administrators. All have different needs and fears. Taverna is designed for expert bioinformaticians, not biologists. Also know the background and experiences of your users. Taverna is propagated by those who do bioinformatics – that is young postdocs and postgraduates. They are familiar with online gaming, instant messaging, file sharing, social networking and internet shopping sites. Hence it was natural to base myExperiment on this experience and others: searching for workflows should be like shopping on Amazon or searching like Google; creating working groups should be like creating groups on Facebook. Familiar interfaces mean no training.

**6. Expect and Anticipate Change.** User needs will change. Success will mean that scientific practice will change, as will expectations. Scientists who were happy with a slow response or minimal data management support now need production level database support and reliable rapid delivery. The choice of pioneering friends will need to change and new classes of user will emerge. When we started out, we replicated current scientific practice. The focus was on interoperating services, making services accessible and workflow creation [6]. Two years later we were inventing new practice: workflows had to integrate data not just link services; discovery had become an issue with so many services; and workflow editing and reuse was as important as creation from scratch. The software

changes the science which changes the software. The challenge for developers is predicting and anticipating these trends, within the six design principles. That's the hard part.

## Reflections

Our principles are based on several years of experience in e-Science, observing both success and failure. In this article we have looked at the design of two quite different pieces of software – the superclient and the Web application. Although they are both in the workflows domain we believe the principles to be more broadly applicable. They are actually about social aspects of software design, which is significant in the collaborative context of e-Science, and they may extend to systems-level science which “integrates not only different disciplines but also, typically, software systems, data, computing resources, and people” [7]. Broader applicability is exemplified by our chemistry colleagues who are ‘blogging the lab’ and have taken a very sympathetic design approach [8].

The realisation of our principles in engineering practice brings tradeoffs. For example, our use of RoR in myExperiment imposed something of a straitjacket which made it difficult for us to ‘act locally’ with multiple user interface models while ‘thinking globally’ in the back end. However it has been hugely effective that we have a programming model that enables us to spend more time with users and less time writing code. We anticipate extensive use of Web 2.0 ‘functionality mashups’ for scientists, and we note that there is considerable consistency between our design and engagement principles and Web 2.0 design patterns.

We have focused here on our principles rather than their implementation. Our management of myExperiment has much in common with Extreme Programming (XP), implemented here in a core team across two sites with one week iterations, on site customers at both and continuous integration. While XP for scientific research has been discussed before [8], we have explored a rather different scenario – decoupled communities of independent and autonomous scientists doing everyday research. We have found it very natural to be agile and responsive with a Web application.

We are both delivering software to scientists and empowering scientists (and their computing experts) to participate in the creation of software – **software is the power behind the scientists, and scientists are the power behind the software**. We believe this empowerment is essential. Being prescriptive will not work – scientists must be as comfortable using their software apparatus as they are any other scientific tools. It's not so much about rolling out software as rolling in users, and it can only happen if the software designers are willing to cross the line and take a user-centric view of the software design process.

## Acknowledgments

The Taverna team, past and present, is extensive: special thanks goes to the lead architect, Tom Oinn, and to Hannah Tipney, our first bio-pioneer. Special thanks to all our developers, PALs, advocates and all our myExperiment ‘friends and family’ including: Jiten Bhagat, Andy Brass, Simon Coles, Don Cruickshank, Paul Fisher, Jeremy Frey, Antoon Goderis, Duncan Hull, Douglas Kell, Matt Lee, Bertram Ludäscher, Danius Michaelides, Stian Soiland, Robert Stevens, Ian Taylor, David Withers and Katy Wolstencroft.

## References

1. Oinn T et al. *Taverna: Lessons in creating a workflow environment for the life sciences*. Concurrency and Computation: Practice and Experience 18(10) pp. 1067-1100, Aug 2006.
2. Fisher P et al. *A systematic strategy for large-scale analysis of genotype–phenotype correlations: identification of candidate genes involved in African trypanosomiasis*, Nucleic Acids Research, 2007, pp. 1–9.
3. De Roure D, Goble CA and Stevens R. *Designing the myExperiment Virtual Research Environment for the Social Sharing of Workflows*. e-Science 2007 – Third IEEE International Conference on e-Science and Grid Computing, 2007. Bangalore, India, 10-13 December 2007. pp. 603-610.
4. Galperin MY. *The Molecular Biology Database Collection: 2007 update*. Nucl Acids Res. 2007 January 12, 2007;35 (suppl\_1):D3-4.
5. Antón AI and Wells D. *Point/Counterpoint: Don't Solve a Problem Before You Get to It / Successful Software Projects Need Requirements Planning*. IEEE Software 20(3), 2003. pp 44-47.
6. Stevens R, Tipney HJ et al. *Exploring Williams-Beuren Syndrome Using myGrid in. Bioinformatics 20:i303-310*. Proc of 12th Intelligent Systems in Molecular Biology (ISMB), 31st Jul-4th Aug 2004, Glasgow, UK.
7. Foster I, Kesselman C. *Scaling System-Level Science: Scientific Exploration and IT Implications*. IEEE Computer Volume 39 (11), Nov. 2006, pp. 31-39.
8. Neylon C, Workshop on Blog Based Notebooks. Abingdon, UK, 28-29 February 2008. See *Science in the Open – An openwetware blog on the challenges of open and connected science*, <http://blog.openwetware.org/scienceintheopen/2008/02/29/>
9. Wood W. A. and Kleb W. L. *Exploring XP for Scientific Research*. IEEE Software 20(3), 2003, pp. 30-36.