Ecological Modelling 221 (2010) 2167–2176

Contents lists available at ScienceDirect

Ecological Modelling

journal homepage: www.elsevier.com/locate/ecolmodel

An IT perspective on integrated environmental modelling: The SIAT case

P.J.F.M. Verweij[a,*], M.J.R. Knapen[a], W.P. de Winter[a], J.J.F. Wien[a], J.A. te Roller[a], S. Sieber[b,1], J.M.L. Jansen[a]

[a] ALTERRA, Wageningen-UR, Droevendaalsesteeg 3, 6708 PB, Wageningen, The Netherlands
[b] ZALF, Leibniz Centre for Agricultural Landscape Research, Eberswalder Straße 84, D-15374, Müncheberg, Germany

ARTICLE INFO

Article history:
Available online 23 February 2010

Keywords:
Software development process
Software architecture
Modelling
Integrated assessment
Assessment tool

ABSTRACT

Policy makers have a growing interest in integrated assessments of policies. The Integrated Assessment Modelling (IAM) community is reacting to this interest by extending the application of model development from pure scientific analysis towards application in decision making or policy context by giving tools a higher capability for analysis targeted at non-experts, but intelligent users. Many parties are involved in the construction of such tools including modellers, domain experts and tool users, resulting in as many views on the proposed tool. During tool development research continues which leads to advanced understanding of the system and may alter early specifications. Accumulation of changes to the initial design obscures the design, usually vastly increasing the number of defects in the software. The software engineering community uses concepts, methods and practices to deal with ambiguous specifications, changing requirements and incompletely conceived visions, and to design and develop maintainable/extensible quality software. The aim of this paper is to introduce modellers to software engineering concepts and methods which have the potential to improve model and tool development using experiences from the development of the Sustainability Impact Assessment Tool. These range from choosing a software development methodology for planning activities and coordinating people, technical design principles impacting maintainability, quality and reusability of the software to prototyping and user involvement. It is argued that adaptive development methods seem to best fit research projects, that typically have unclear upfront and changing requirements. The break-down of a system into elements that overlap as little as possible in features and behaviour helps to divide the work across teams and to achieve a modular and flexible system. However, this must be accompanied by proper automated testing methods and automated continuous integration of the elements. Prototypes, screen sketches and mock-ups are useful to align the different views, build a shared vision of required functionality and to match expectations.

1. Introduction

The last three decades the environmental modelling community has developed numerous models (Reynolds and Acock, 1997; Papajorgji et al., 2004). These modelling efforts have evolved from single disciplinary to interdisciplinary models "to allow for a better understanding of complex phenomena enabling the evaluation of the whole cause effect chain from a synoptic perspective by combining, interpreting and communicating knowledge from diverse scientific disciplines" (Rotmans and Dowlatabadi, 1998). Integrated Assessment Modelling (IAM) simulates both the natural and socio-economic systems in applications like scenario analysis and evaluation of the environmental, economic and social consequences of different policy strategies (Parker et al., 2002; Van de Sluijs, 2002).

Policy makers have a growing interest in integrated assessments of policies (Van Ittersum and Brouwer, 2009) on which the IAM community is reacting by extending the application of model development from pure scientific analysis towards application in decision making or policy context (Matthies et al., 2007; Sterk et al., 2009).

Typically many individuals from different institutions, diverse background and roles are involved in the development of an IAM (Hinkel, 2009), modellers, indicator experts, domain experts, tool users, software engineers, managers and donor representatives, resulting in as many views on the proposed tool which especially in the early phases are not always exactly envisioned. Dissenting views may continue to exist unnoticed when design is not made concrete from the beginning. Even during the development advanc-

* Corresponding author at: Tel.: +31 317 481601; fax: +31 317 489000.
E-mail address: peter.verweij@wur.nl (P.J.F.M. Verweij).
[1] Current address: European Commission, Joint Research Centre (JRC), Institute for Prospective Technological Studies (IPTS), Edificio Expo, Avda, Inca Garcilaso s/n, 41092 Seville, Spain.

0304-3800/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.ecolmodel.2010.01.006

ing research continues to lead to an improved understanding of the system. Therefore, early specifications tend to be altered later on. Accumulation of changes to the initial design obscures the design, usually vastly increasing the number of defects in the software (Larman, 2004).

Initial IAM was targeted at the development of comprehensive integrated systems, like the RAINS model (Alcamo et al., 1990), or IMAGE model (Rotmans, 1990). Current IAM development focuses at the modelling itself e.g., steps to develop a model (Jakeman et al., 2006); participatory modelling (Voinov and Gaddis, 2008); quality assurance in modelling (Scholten et al., 2007), or on modularity to allow configuration in accordance with the question at hand (Reynolds and Acock, 1997; Donatelli et al., 2002; Gijsbers et al., 2002; Argent, 2004; Leimbach and Jaeger, 2004; Papajorgji et al., 2004; Hinkel, 2009). Although IAM models are implemented through software, IAM seems to make little use of software engineering methodologies. The software engineering community uses concepts, methods and practices to deal with ambiguous specifications, changing requirements and incompletely conceived visions, and to design and develop maintainable/extensible quality software while safeguarding usability aspects.

This paper aims to introduce environmental modellers to software engineering concepts and methods which have the potential to improve model and tool development. Experiences with the development of the Sustainability Impact Assessment Tool (SIAT) will serve as an illustrative case study.

Section 2 introduces some important software engineering concepts and methods which can have a large effect on software quality and are easy to implement. All of these concepts and methods were used for the development of SIAT as explained in Section 3. Finally, Section 4, discusses what has been learned by applying the software engineering concepts and methods for the development of SIAT, confronts it with literature and concludes by explaining its added value to IAM development in general.

2. Software engineering methods and concepts

Software engineering is a field of study concerning the application of a systematic and disciplined approach for the development, operation and maintenance of complex software (Abran and Moore, 2004). Main clusters of interest are: (i) the process – how to get from system requirements to a product; (ii) structure – the design of the system; (iii) technology – what technology will be (re)used, and; (iv) organization – assign tasks to responsible individuals and/or organizations. Software quality assurance (Srivastava and Kumar, 2009) intersects with all clusters.

IAM is at an early stage of applying software engineering principles. The following paragraphs introduce elementary methods and concepts which can have a large effect on quality and are easy to implement.

2.1. Software development methodology

A common metaphor for software engineering is construction. This metaphor works out well when all requirements can be specified upfront in detail. Typically in research projects requirements are not clear from the beginning. Here the gardening metaphor from Hunt and Thomas is more suitable (Hunt and Thomas, 1999). Constant work is needed to keep it in the required shape. Choosing the right development process is a critical success factor to the development and use of a software system.

A software development methodology is a prescriptive model that establishes the order in which a project specifies, prototypes, designs, implements, reviews, tests and performs its activities. It primarily exists to co-ordinate people involved in the development of the software (Cockburn, 2000): architects, designers, implementers, testers, users, researchers and project co-ordinators. Literature gives us many development methods to choose from, varying from the formal Rational Unified Process (Kruchten, 2003) and strictly phased waterfall method (Royce, 1970) to highly adaptive agile methods like eXtreme Programming (Beck and Andres, 2004), SCRUM (Schwaber and Beedle, 2001), or Chrystal (Cockburn, 2004). Agile methods demand to get continuous user feedback during short design-implement-test-deliver iterations.

Which method to choose depends on: (i) understanding of system requirements and the ability to update them during project execution; (ii) software development expertise; (iii) team size and team distribution; (iv) decision making, leadership and culture; (v) necessity to have visual presentations before the end of the project, either for customers, or management; and (vi) predefined schedule constraints (McConnell, 1996; Cockburn, 2000; Tate, 2005; Poppendieck and Poppendieck, 2006).

2.2. Domain analysis

A common language and a shared understanding of the application context by all stakeholders is crucial as this is the basis for further analysis. The design of a software product starts therefore by analysing the *conceptual domain* to which the software applies. A *conceptual domain analysis* yields common grounds for further specific analysis (Champeaux et al., 1993) by identifying, collecting, organizing, and representing the relevant information in a domain, based upon the study of knowledge captured from users and domain experts by means of workshops and interviews; underlying theory in literature; and the study of existing systems within the domain. Domain analysis carefully delineates the domain being considered, organizes an understanding of the relationships between the various elements in the domain, considers commonalities and differences of the systems in the domain and represents this understanding in a useful way (Nilsen et al., 1994). Result of the analysis is a *domain model*: a simplified, abstract image of reality. In the analysis notions from the domain and relations between those notions are described.

2.3. Usability and prototyping

A broader scope and applicability can be achieved when an assessment tool is targeted at the less technical experienced user (Matthies et al., 2007). Within the User Centered Design approach (Raskin, 2000) usability requirements drive the features and technical development by studying the usefulness with the intended users. Central usability characteristics include: learnability, efficiency, memorability, low error rate and satisfaction of user experiences when working with the software (Nielsen, 1992; Holzinger, 2005).

Prototypes of an interface design can be used to test usability with users. Holzinger (2005) gives an overview on methods to inspect and test usability aspects with prototypes. Prototypes can be incomplete versions of the software product, but may as well be screen designs in a software presentation tool, or even hand drawn sketches on paper (Sefelin et al., 2003). They allow users to evaluate developers' proposals for the interface construction of the product by actual testing, rather than having to interpret and valuate the design based on descriptions. The main objective of a prototype is to find out if the developers are on the right track and to further feed requirement discussion.

Prototypes are also useful to test technical issues, such as performance, interfacing between components and service availability. In general a prototype is an inexpensive way to try out ideas so that as many issues as possible are understood before the real implementation is made (Tate, 2005).

2.4. Architecture

The increasing size and complexity of software force the use of abstraction and to break the system down into separate elements of concern in which each element has its own functional responsibilities. Such a common abstraction of a system, or architecture, manifests early design decisions through which the system to be build can be analysed. As such an architecture helps communication among stakeholders as a basis for mutual understanding, negotiation and consensus by documenting system qualities, like modularity, adaptability, extensibility, maintainability and portability. Through an architecture the system can be compared with others, reusable components can be located and cost estimates can be made. Understanding of and consensus on an architecture is important as it defines early design decisions which are hardest to change and therefore most critical to get right (Bass et al., 2003).

From the many existing software architectures there is one that is very often used for many applications: the *layered architecture*. In the layered software architecture the system is split up into a number of layers in which each layer can be built, tested, changed and reused independently. As a general rule each layer only has dependencies on those below it, limiting the effect of changes and thereby increasing maintainability. For instance the layers of 5-layered architecture consist of: (1) *presentation*, user interface; (2) *application*, workflow, e.g. what screen appears when a certain button is pressed, or enabling of controls when login is valid; (3) *services* for controlling transactions; (4) *domain*, program logic representing domain knowledge; and (5) *persistence* for storing state.

The different layers may be running on a single computer, but can also be divided on separate machines. For example the *presentation* and *application* layer could be running as a web-client and the *persistence* layer can be implemented by a relational database running on a separate server. This is referred to as a multi-tier architecture.

2.5. Quality

Many aspects co-determine the quality of software (McConnell, 2004). Full elaboration of the subject is beyond the scope of this paper. Reusability of components or services, possibly based on standards (Krueger, 1992; Simcoe, 2006), is mentioned as quality characteristic in the context of ecological and agricultural modelling (Reynolds and Acock, 1997; Donatelli et al., 2002; Papajorgji et al., 2004; Holzworth et al., 2010). Two other quality aspects are detailed here and are easy to implement into the development process and have a profound effect on the defect rate, coding efficiency, the understandability and the spread of knowledge of the code.

Like scientific papers, source code quality is improved by reviews. The most intensive form of code reviewing is pair programming, where one developer continually monitors another developer that is entering the code (Beck and Andres, 2004). A large number of defects and small-scale design flaws are intercepted this way before they become part of the standing code base. The code tends to be better readable, understandable, and maintainable. While at first sight it may seem unproductive to have two developers producing the code, the early interception of defects saves much time later that needs not be spent on finding and solving defects and refactoring weak design choices. Even more important is the reduction of the number of effects that otherwise would remain unnoticed unto the final product (McConnell, 1996; Tate, 2005).

In addition to code reviews, correct functioning of code can be guaranteed by accompanying all production code by unit tests. These tests can be run automatically and are to test various kinds of foreseen and unforeseen calls to the code against a predicted result. The collection of unit tests builds up during the entire development period and can be run any moment to check whether new alterations may have unwanted side effects to existing code. Seemingly complex code defects can be automatically traced back to simpler underlying code. Hence the extra time invested in writing unit tests saves time solving unwanted side effects.

3. SIAT development

3.1. What is SIAT

SIAT is a web application to estimate the possible consequences of different policy assumptions on multifunctional land use and its sustainability within different images of the future (Verweij et al., 2009). SIAT (Fig. 1) allows the user to identify those geographical areas that are most sensitive to particular policies, identify regional differences and analyse causes, look at potential 'trade-offs' and undertake all analysis dynamically (Potschin and Haines-Young, 2008).

SIAT was developed within the integrated research project entitled 'Sustainability Impact Assessment: Tools for Environmental, Social and Economic Effects of Multifunctional Land Use in European Regions (SENSOR)' funded through the EU Framework Programme 6. The project covered the sectors forestry, nature conservation, agriculture, energy, transport and tourism (Helming et al., 2008).

SIAT uses a model chain to translate policies together with certain images of the future into impacts (Fig. 2a). These drivers are translated into land-use changes from which in turn in combination with constant-factor maps social, economic and environmental indicators are derived. Constant-factor maps contain parameters which are expected to be constant throughout the modelling. Finally, regional sustainability limits for the indicators by means of land-use functions are assessed (Pérez-Soba et al., 2008; Paracchini et al., in press).

Land-use changes are determined by the use of a modelling framework (Jansson et al., 2008) including the macro-econometric model NEMISIS (Brecard et al., 2006), the forestry model EFISCEN (Nabuurs et al., 2001; Schelhaas et al., 2007), the agricultural model CAPRI (Britz et al., 2003; Britz and Witzke, 2008) and the land-use allocation model DYNA-CLUE (Verburg et al., 2004). Since the modelling framework was complex to work with and took a long time to calculate impacts of various policies SIAT was planned to use a meta-model (Sieber et al., 2008) derived from the modelling framework (Kuhlman, 2008).

New policies, baselines, target times and indicators at different spatial extent and spatial resolution can be added to SIAT making it adaptable to future applications.

3.2. Development process

During workshops, meetings and interviews with scientists from within the project, policy makers at EU level and regional stakeholders we disseminated ideas and received feedback that further specified system requirements. Iteratively the same group of scientists and policy makers and different groups of regional stakeholders were contacted during the full four and half year project duration. The presentations and discussions in the workshops were structured around definitions of policies, impact comparisons and visualizations, and the need of explaining assumptions and the causal relation between drivers and impacts.

Initially several detailed interpretations of the system were presented. A common vision and integrating concept, however, did not arise until the presentation of a first prototype that was developed together with a graphical designer (Verweij et al., 2006). This prototype provoked adequate feedback that helped to stabilize the

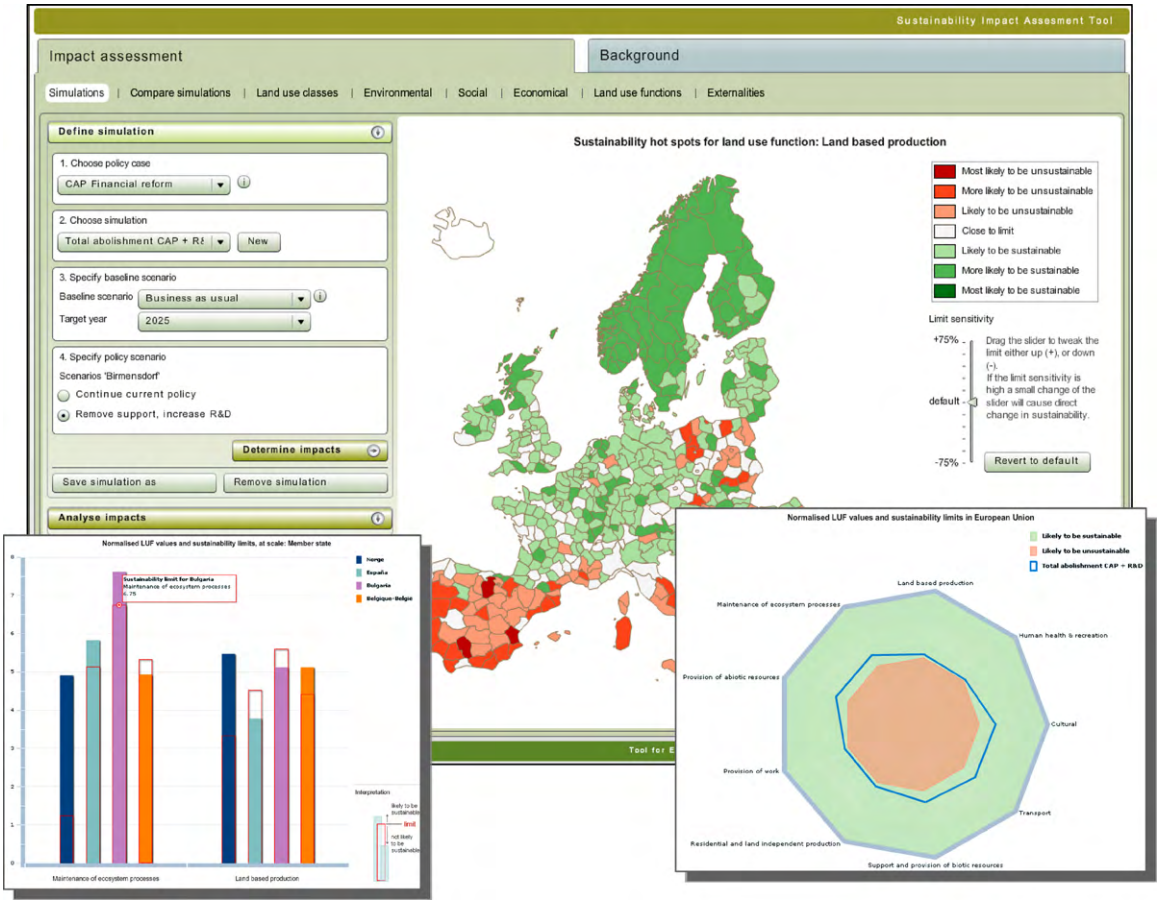

Fig. 1. SIAT user interface impression. The main screen shows the definition of a policy and a map depicting regional differences of the land use function 'land based production'. In the radar chart in the inset on the bottom-right, all land use functions' scores in relation to the sustainability limit (edge of the red circle) are drawn allowing to find trade-offs. In the chart in the inset on the bottom-left, bars are drawn representing scores of 2 land use functions and 4 member states. Bars are grouped by land use function. The thin red lines plotted on top of the bars represent limits which vary per land-use function and member state.

conceptual domain model and gave direction to system development (Fig. 3).

The domain model (Fig. 4) shows that an integrated assessment compares indicators in different images of the future that are influenced by several drivers. Drivers have been divided between those that can be affected by policy versus external drivers such as climate change, technological innovations and world population.

Based on the feedback a multi-tier architecture was selected, in which the models and visual representations of impacts find a place as exchangeable components. Like all other system components, in the agile development process that has been used, the design of the system architecture was not rigidly fixed, but subject to change whenever required. However, since the architecture is fundamental to the application, major changes at this level were less likely to occur.

To speed up development we searched for reusable components and services with a strong preference for open standards, such as the OpenMI to provide a standardized interface to describe, link and transfer data between models on a time step basis (Moore and Tindall, 2005) and Web Mapping Service to produce maps of spatially referenced data dynamically from geographic information (OGC, 2004). Components and services for architecture were searched for during its design, while others were searched for as required at any stage in the development. Examples of the latter

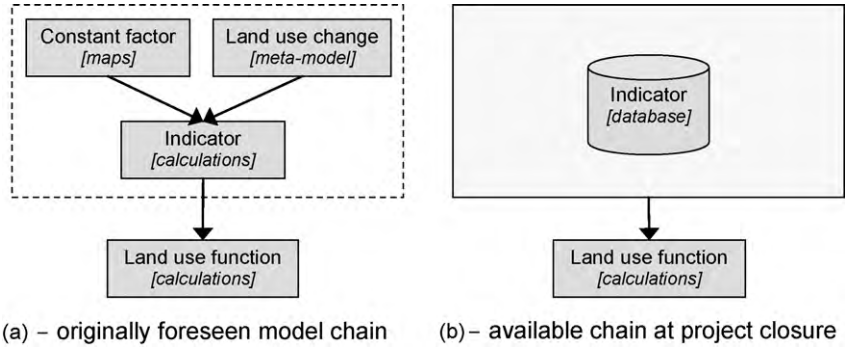

Fig. 2. Model chain. Model components making up the model chain use a standardized interface making them (technically) interchangeable. This is the case between figure a and b in which three model components are replaced by one with equal behaviour. (a) Originally foreseen model chain and (b) available chain at project closure.

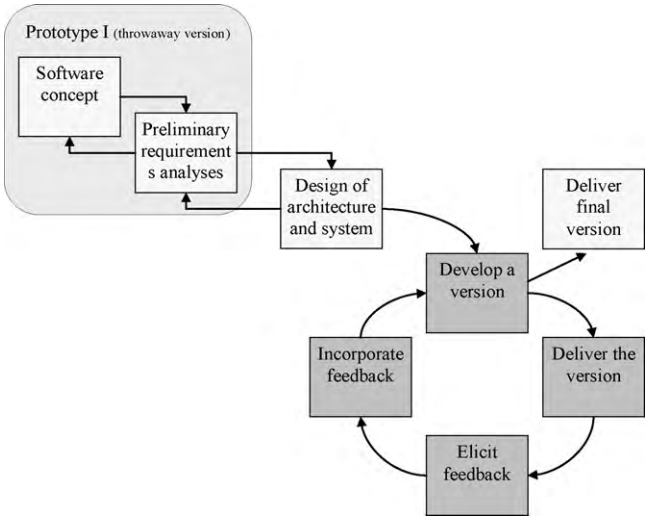

Fig. 3. Evolutionary development methodology (after (McConnell, 1996)). Project activities are located in the boxes while the arrows establish the order in which the activities are carried out. The box with rounded corners delimits the activities resulting in SIAT prototype I.

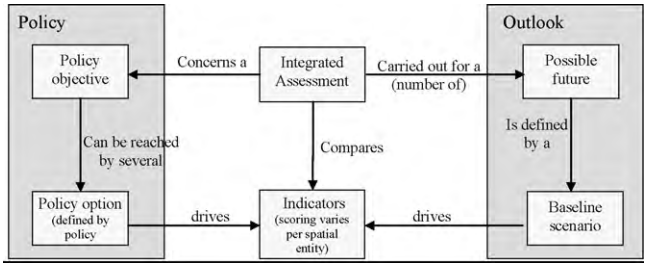

Fig. 4. SIAT domain model. SIAT estimates the possible consequences (quantified by indicators) of different policy options within different images of the future.

category include graph visualization components, or data parsing components.

Usability testing included *storyboards*, a sequence of *screen sketches* on how we understood the targeted impact assessment tool, how it should/could look and was to be interacted with (Fig. 5).

At the same time we gathered feedback creating new *story cards*, a short formulation of a user requirement (Beck and Andres, 2004), to be implemented in a following iteration (Fig. 6). A story card was implemented vertically through all architectural layers in contrast to the more classical approach of developing layer-by-layer horizontally.

Iterations implemented as many story cards as would fit in a time frame of 3 weeks. Iterations resulted in an operational new release, on which user feedback and new story cards or new prioritization of story cards were based. Story cards were stacked with the highest priority card on top. Three releases have been tagged as prototypes for formal project deliverables.

Within every iteration, source code was added to and changed within the existing code base. Unit tests were used to ensure proper functioning of previously developed code. Refactoring (Fowler et al., 1999) was applied liberally to make the internal structure of software easier to understand and cheaper to modify without changing its observable behaviour. All code and unit tests were checked into a version control system on a daily basis.

Software development took place in two teams based at separate locations and organizations. One team worked on a single model component without formal development method. The larger team consisted of 4–6 software developers in one room and used the development method 'evolutionary development' using story cards. Implementing a story card started with the design and breakdown into smaller tasks. Day tasks were assigned each morning during a 20 min stand-up meeting in which each developer also shortly reported on the progress made the previous day. Story cards were assigned starting with the highest priority. The implementation of story cards was done using pair programming assuring

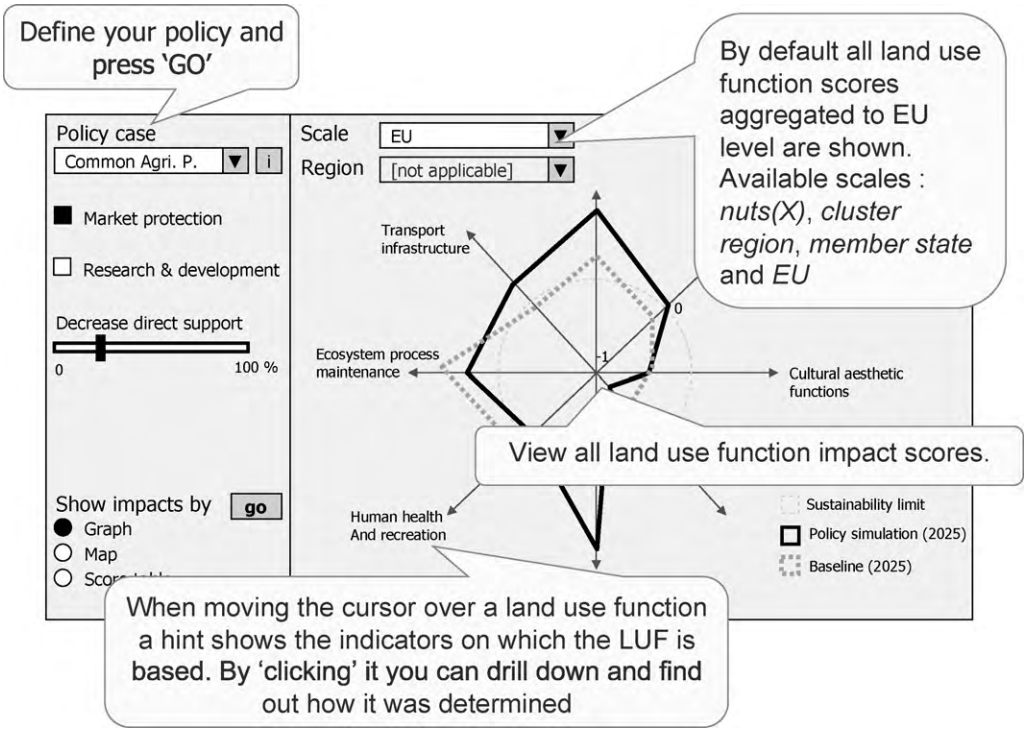

Fig. 5. Screen sketch example. A Graphical User Interface sketch showing all controls in an anticipated state. Major points of interest are explained in text balloons.

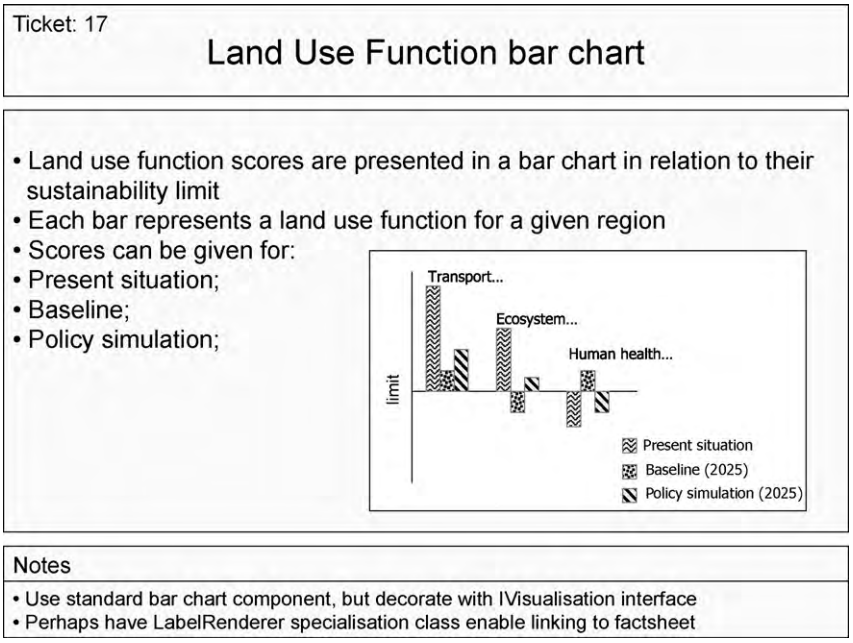

Fig. 6. Story card example. Three elements make up the story card: (i) a unique identifier and a concise (but possibly improved) title; (ii) a short formulation of a requirement in user terminology which may be supported by a sketch; and (iii) developer notes.

continuous code review by the pair partners. The composition of the pairs was reshuffled after each card completion.

3.3. Results

Using the 5-layered architecture (Section 2.5) the SIAT system was broken down into elements that overlap as little as possible in features and behaviour. This principle of Separation of Concerns accommodated the organizational structure of the SENSOR project in which different engineering groups were responsible for development of different software elements. Fig. 7 shows the elements which make up the total SIAT system and their relation to each other:

- Graphical User Interface (*GUI*);
- *Simulation services* – access to the domain model such as available policy instruments, indicators, spatial divisions, meta-information;
- *Models* – are wrapped in a standardized interface using the *OpenMI* (Moore and Tindall, 2005) which supports to use modular model compositions;
- *Map services* – gives access to geo-referenced images representing maps (WMS: OGC, 2004);
- *Factsheet service* – resources containing fact sheets in xml format;
- *3D landscape visualization services* – provides 3D landscape images showing how the landscape might look in a modelled future (Snizek et al., 2008).

Each element can be located at a different physical location. For practical reasons such as maintenance, performance, or security

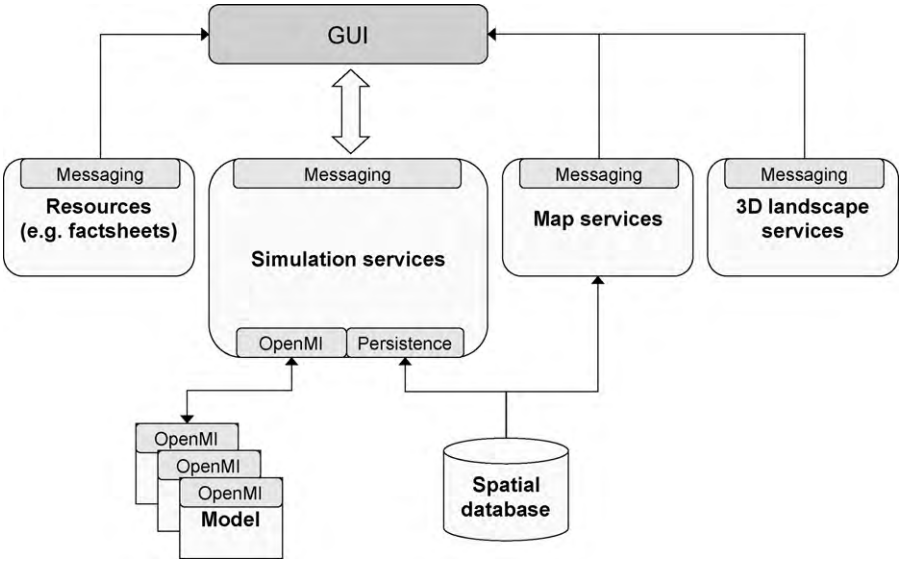

Fig. 7. SIAT system architecture. Each box represents an element within the system. The arrows between the elements indicate the communication direction between them. The Graphical User Interface (*GUI*) element resides at the client's computer, while all others are located on the server. Models are wrapped using a standardized interface (OpenMI) facilitating the formation of a modular model composition.

this would not be optimal. The *GUI* runs at the client's machine in a web browser. The larger part of the systems' elements reside on a single web server while the database is located on another server within the same domain. The 3D visualization services are hosted by another organization.

The architecture is data driven, in terms of policy instruments, baseline scenarios, spatial divisions, indicators, documents, geo-related data (e.g. climate zones), etc. Models are substitutable, i.e. they can be substituted with other implementations without the need to re-engineer the entire software system. The core system services are loosely coupled to the (web based) user interface, facilitating the production of specific front-ends interacting with the same back end.

The 3D landscape visualization and the meta-model implementations were excluded from the latest release due to instability. A pre-calculated (result-based) component model replaced the originally planned regression based meta-model (Fig. 2b). The decision not to release SIAT with the meta-model component could be postponed until late in the project without running into problems as a consequence of the open model architecture.

4. Discussion

4.1. Agile development method

Agile development methods were used in the implementation of the SIAT tool. These were most suitable, as the outcome of research projects are to some extent unpredictable due to their innovative nature. Unpredictable outcomes lead to ambiguous and changing specifications of what SIAT should do. Part of this unpredictable outcomes are caused by advances in scientific understanding of the domain. Generally speaking these unpredictable outcomes make it hard to direct the development from the onset of the project and use exhaustive upfront design phases. Instead, the development process must be agile and quickly adjust to changing or new insights from the rest of the research project, resulting in modified specifications. Agile processes are among the top ten key factors that determine the success of software projects (Standish Group, 2005).

A selection (e.g. story cards, pair programming, daily stand-ups) of best-practices from agile methods was chosen, based on the literature and personal preference. The software development was based on short responsive iterations, in which the specifications of the SIAT tool were adjusted to the lessons learned. Each development iteration consisted of short design, implementation, testing and release-phases and lasted no longer than 4 weeks. Other parts of the research project operated on longer iterations. The use of these agile practices helped to match the results of the software development with the results from the others domains in the research project and to manage expectations of users with respect to tool functionality.

The agile approach was challenged, however, by differences in culture between the two teams. One team operated more on the basis of authorized craftsmanship, which implies a focus on delivering a functional tool instead of documenting a semi-functional tool and is based on trust in the software development team. The other team followed a more procedural approach to software development, which is advisable when the employing organization prefers clear policies and procedures (Boehm and Turner, 2004). The team using the agile practices developed the major part of the tool, while the other team developed a replaceable component with a clear interface. The clear interface resolved the conflict in development methods, and allowed the teams to cooperate. The parallel use of different development methods was only possible in this case, as the definition and interface of one component was rigidly fixed at the start of the development process. If it is not possible to rigidly define these components and their interfaces, it is advisable to use agile methods in all cases and not trying to mix methods.

Literature (Salo and Abrahamsson, 2008) reports the increased development speed and improved code quality characteristics of agile methods compared to formal methods, which was not experienced at the start of the development process of SIAT. Initially there were fluctuations in development speed and quality, due to team members working on all layers of the architecture. This typically required team members to learn new skills, for example a database developer learning about the user interface. This learning process was facilitated through the use of pair programming. Later on in the development process, the use of agile methods resulted in a shared understanding of the system between team members and a stable and productive development pace.

4.2. Continuous integration and separation of concerns

SIAT was divided in tiers (e.g. presentation, model, data) and components (e.g. meta-model, map services) according to the principle of Separation of Concerns. This enabled us to distribute the workload over two development teams. Both teams could progress at their own pace, however some safeguards were needed to prevent integration problems, because of changing component interfaces during development by one team.

The selected integration approach for components and tiers during the development process was the use of proxies. Most components were initially represented by proxies, i.e. simple substitutes that display the same behaviour externally, but not necessarily produced semantically valid data. Proxies allow gradual implementation of functionality of a component and at the same time have a working end-to-end system. A drawback of the use of proxies is that component interface changes during the development phase can easily be made and not be detected until the integration phase, as multiple components often are maintained separately. The integration phase is close to a milestone or deadline and problems in this phase can be costly and stressful to solve. Solving such integration problems could profit from continuous integration as utilized in the SEAMLESS-Integrated Project (Van Ittersum et al., 2008; Wien et al., 2010). Continuous integration means semantically and technically validating the integration of each of the components and tiers into the whole. It is a software development practice where team members integrate their work (at least) daily. Each integration is verified by automated tests and automated builds (Fowler, 2006).

Automated unit testing provides trust in the proper functioning of the code base when all tests pass. Ideally all methods of all classes are verified through unit tests. We built unit tests for the *simulation services* and the OpenMI compliant *models*. Incremental development and refactoring of these elements were safe as errors could be quickly identified and fixed. For the *GUI* we did not use unit tests as we found the technology of the testing framework immature and burdensome to work with. As the *GUI* grew in time and became in need of refactoring this was understood as a wrong decision. Even some simple errors were time consuming to locate. As it is advisable to write unit tests for source code of domain or application tiers, unit tests must also be available for source code that is part of the presentation tier (i.e. *GUI*). Fortunately, testing frameworks for presentation tiers are still improving.

As part of the separation of concerns, SIAT strictly separates subject matter (data, or a modular model) from program logic permitting changes in application behaviour. This implies that when adding additional policy instruments, indicators, or spatial divisions to the database, they will become available through the system without the need to recompile. Data deliveries were often late due to reservations researchers have to deliver premature data. Since no assumptions were made about the contents of the data in

the program code, work on it could continue independently, without interfering with the data production. Nevertheless it must be stressed that early integration of data and application is important to check the relations and format of the data against the program logic. In addition, such a separation of data and logic facilitates the application of the system in areas where it was not originally developed for. Or, with substitutable models, specific model components could be exchanged with versions suitable to the new application.

Originating from the water domain the OpenMI was successfully used for linking models in SIAT and in other environmental applications (Janssen et al., 2009; Lindner et al., this issue). By using OpenMI we were able to have an alternative component act as substitute for the ones that could not be realized. An extension to the standard of OpenMI for the description and exchange of non-numerical, or complex data simplified the application for SIAT models (Verweij et al., 2007; Wien et al., 2010). Knapen et al. (2009) describe the use of ontologies (Villa et al., 2009) for describing exchangeable information with OpenMI components. The architecture helped us to find similarities with the SEAMLESS-IF tool (Wien et al., 2010) providing a basis for cooperation on technical design and implementation.

4.3. Users and usability

Clear objectives for tool development are not obvious since often various stakeholders have different goals. A concise and comprehensive vision statement that is agreed upon or at least acknowledged by the stakeholders is necessary to ensure a successful tool development. Furthermore, user involvement is prerequisite for directing the development process (Mysiak et al., 2005; Standish Group, 2005; Jakeman et al., 2006; McIntosh et al., 2008) and raising and balancing expectations (Sterk et al., 2008) throughout the project.

Like the first version of mDSS (Mysiak et al., 2005) we focused on the *GUI* while using dummy data and models for illustration. Rizzoli and Young (1997) distinguish three main categories of users: the scientist, the manager/policy maker and the stakeholder. We targeted SIAT at the last two categories and started looking for examples in a similar domain such as the unpublished 'ATEAM mapping tool' and EURURALIS (Westhoek et al., 2006). The EURURALIS tool was appreciated for the visualization of output helping users to get an improved understanding on interdependencies (Sterk et al., 2009). The assignment of user roles to virtual user representatives at the EU level as assessed in several workshops (Tabbush et al., 2008) provided an opportunity to disseminate ideas while real user representatives were selected from the project participants for attaining the required development direction.

Uncertainty on the user type targeted with the SIAT tool during the early project stages made us decide to develop a throw away prototype as a starting point. Argent and Grayson (2001) suggest that such an initial system helps users to overcome difficulties to explicitly express what they expect from a system. As such the throw away prototype was invaluable to arrive at a shared and explicit vision and receive detailed feedback on software requirements. Subsequent incremental releases ensured that software development stayed close to expectations based on the shared vision of the end-product.

As part of the development of the throw away prototype and releases, screen sketches and story boards depicting *GUI* suggestions were drawn. These can be drawn fast in comparison to implementing them in actual software and are far less ambiguous than textual suggestions without pictures. Participatory design of screen sketches was found especially effective in small workshops where participants drew jointly on a whiteboard while discussing their design.

All SIAT versions have benefited from the participation of a graphical designer for drawing sketches during the development. Literature highlights the importance of design to remove barriers, which results in an increasing willingness to use a tool (Lu et al., 2001). Tractinsky et al. (2000) even states that a well designed product is a better usable product and the SIAT tool attracted positive feedback with its professional appearance.

5. Conclusion

This paper introduces some common software engineering principles and demonstrated their usefulness through an application of these principles to the SIAT tool development. As a software development method, agile practices seem to best fit research projects, that typically have unclear upfront and changing requirements and many different views on the functionality. Separation of concerns helps to divide the work across teams participating in the research project and to achieve a modular and flexible system. However, this must be accompanied by proper automated testing methods and continuous integration. To align the different views on functionality prototyping, screen sketches and mock-ups can be used to build a shared vision of required functionality and to match expectations.

Acknowledgements

The authors acknowledge the contributions of the EU Framework Programme 6 funded integrated projects SENSOR and SEAMLESS, Janneke Roos Klein-Lankhorst, Sander Janssen, the guest editor Martin K. van Ittersum and an anonymous reviewer.

References

Abran, A., Moore, J.W., 2004. Guide to the Software Engineering Body of Knowledge. IEEE, http://www.computer.org/portal/web/swebok.
Alcamo, J., Shaw, R., Hordijk, L., 1990. The RAINS Model of Acidification: Science and Strategies in Europe. Kluwer, Dordrecht.
Argent, R.M., 2004. An overview of model integration for environmental applications—components, frameworks and semantics. Environ. Model. Software 19, 219–234.
Argent, R.M., Grayson, R.B., 2001. Design of information system for environmental managers: an example using interface prototyping. Environ. Model. Software 16, 433–438.
Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice, second edition. Addison-Wesley.
Beck, K., Andres, C., 2004. Extreme Programming Explained: Embrace Change. XP. Addison-Wesley Professional, 224 pp.
Boehm, B., Turner, R., 2004. Balancing Agility and Discipline—A Guide for the Perplexed. Addison-Wesley, 266 pp.
Brecard, D., Fougeyrolles, A., Mouel, P.L., Lemiale, L., Zagame, P., 2006. Macro economic consequences of European research policy: prospects of the Nemisis model in the year 2030. Res. Policy 35, 910–924.
Britz, W., Witzke, P., 2008. CAPRI Model Documentation 2008: Version 2. http://www.capri-model.org/docs/capri_documentation.pdf.
Britz, W., Perez, I., Wieck, C., 2003. Mid-Term Review Proposal Impact Analysis with the CAPRI Modelling System. Mid-Term Review of the Common Agricultural Policy.
Champeaux, D., Lea, D., Faure, P., 1993. Object Oriented System Design. Addison-Wesley, 532 pp.
Cockburn, A., 2000. Selecting a project's methodology. IEEE Software, 64–71.
Cockburn, A., 2004. Crystal Clear: A Human-Powered Methodology for Small Teams. Agile Software Development Series. Addison-Wesley Professional, 336 pp.
Donatelli, M., Van Ittersum, M.K., Bindi, M., Porter, J.R., 2002. Modelling cropping systems—highlights of the symposium and preface to the special issues. Eur. J. Agron. 18, 1–11.
Fowler, M., 2006. Continuous integration. http://martinfowler.com/articles/continuousIntegration.html.
Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D., 1999. Refactoring: Improving the Design of Existing Code. The Addison-Wesley object technology series. Addison-Wesley, 464 pp.
Gijsbers, P., Moore, R., Tindall, C., 2002. Towards OMI an open modelling interface and environment to harmonise european developments in water simulation software. In: Cluckie, I.D., Han, D., Davis, J.P., Heslop, S. (Eds.), Hydroinformatics. IWA Publishing, Cardiff, pp. 1268–1275.
Helming, K., Tscherning, K., Koenig, B., Sieber, S., Wiggering, H., Kuhlman, T., Wascher, D., Perez-Soba, M., Smeets, P., Tabbush, P., Dilly, O., Huettl, R., Bach, H., 2008. The SENSOR approach. In: Helming, K., Tabbush, P., Perez-Soba, M. (Eds.), Sustainability Impact Assessment of Land Use Changes. Springer.

Hinkel, J., 2009. The PIAM approach to modular integrated assessment modelling. Environ. Model. Software 24, 739–748.
Holzinger, A., 2005. Usability engineering methods for software developers. Commun. ACM 48, 71–75.
Holzworth, D.P., Huth, N.I., De Voil, P.G., 2010. Simplifying environmental model reuse. Environ. Model. Software 25, 269–275.
Hunt, A., Thomas, D., 1999. The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley, 320 pp.
Jakeman, A.J., Letcher, R.A., Norton, J.P., 2006. Ten iterative steps in development and evaluation of environmental models. Environ. Model. Software 21, 602–614.
Janssen, S., Athanasiadis, I., Bezlepkina, I., Knapen, M.J.R., Li, H., Perez Dominguez, I., Rizzoli, A., Van Ittersum, M.K., 2009. Linking models for assessing agricultural land use change, PhD thesis, Wageningen University, Wageningen (Chapter 6).
Jansson, T., Bakker, M., Boitier, B., Fougeyrollas, A., Helming, J., Van Meijl, H., Verkerk, P., 2008. Cross sector land use modelling framework. In: Helming, K., Tabbush, P., Perez-Soba, M. (Eds.), Sustainability Impact Assessment of Land Use Changes. Springer, pp. 159–180.
Knapen, M.J.R., Athanasiadis, I., Jonsson, B., Huber, D., Wien, J.J.F., Rizzoli, A., Janssen, S., 2009. Use of OpenMI in SEAMLESS. In: Van Ittersum, M.K., Wolf, J., van Laar, G. (Eds.), Conference on Integrated Assessment of Agriculture and Sustainable Development: Setting the Agenda for Science and Policy (AgSAP 2009). Egmond aan Zee, The Netherlands.
Kruchten, P., 2003. The Rational Unified Process: An Introduction. Object Technology. Addison-Wesley, 310 pp.
Krueger, C., 1992. Software reuse. ACM Comput. Surv. 24, 131–183.
Kuhlman, T., 2008. Scenarios-driving forces and policies. In: Helming, K., Tabbush, P., Perez-Soba, M. (Eds.), Sustainability Impact Assessment of Land Use Changes. Springer, pp. 131–157.
Larman, C., 2004. Agile & Iterative Development—A Managers Guide. Agile Software Development Pearson Education, Boston, US, 342 pp.
Leimbach, M., Jaeger, C., 2004. A modular approach to integrated assessment modeling. Environ. Model. Assess. 9, 207–220.
Lindner, M., Suominen, T., Palosuo, T., Garcia-Gonzalo, J., Verweij, P., Zudin, S., Paivinen, R., this issue. ToSIA – a tool for sustainability impact assessnebt of forest-wood-chains. Ecol. Model.
Lu, H.-P., Yu, H.-J., Lu, S.S.K., 2001. The effects of cognitive style and model type on DSS acceptance: an empirical study. Eur. J. Oper. Res. 131, 649–663.
Matthies, M., Giupponi, C., Ostendorf, B., 2007. Environmental decision support systems: current issues, methods and tools. Environ. Model. Software 22, 123–127.
McConnell, S., 1996. Rapid Development: Taming Wild Software Schedules. Microsoft Press Redmond, WA, USA, 647 pp.
McConnell, S., 2004. Code Complete. Microsoft Press, Redmond, USA, 156 pp.
McIntosh, B.S., Giupponi, C., Voinov, A., Smith, C., Matthews, K.B., Monticino, M.J., Kolkman, M.J., Crossman, N., Van Ittersum, M.K., Haase, D., Haase, A., Mysiak, J., Groot, J.C.J., Sieber, S., Verweij, P., Quinn, N., Waeger, P., Gaber, N., Hepting, D., Scholten, H., Sulis, A., Van Delden, H., Gaddis, E.J.B., Assaf, H., 2008. Bridging the Gaps Between Design and Use: Developing Tools to Support Environmental Management and Policy Developments in Integrated Environmental Assessment, Elsevier, pp. 33–48.
Moore, R., Tindall, C., 2005. An overview of the open modelling interface and environment (the OpenMI). Environ. Sci. Policy 8, 279–296.
Mysiak, J., Giupponi, C., Rosato, P., 2005. Towards the development of a decision support system for water resource management. Environ. Model. Software 20, 203–214.
Nabuurs, G., Päivinen, R., Schanz, H., 2001. Sustainable management regimes for Europe's forests – a projection with EFISCEN until 2050. Forest Policy Econ. 3, 155–173.
Nielsen, J., 1992. The usability engineering life cycle. Computer 25, 12–22.
Nilsen, R., Kogut, P., Jackelen, G., 1994. Component Provider's and Tool Developer's Handbook Central Archive for Reusable Defense Software (CARDS). STARS Informal Technical Report Unisys Corporation.
OGC, O.G.C.I., 2004. Web Map Service.
Papajorgji, P., Beck, H.W., Braga, J.L., 2004. An architecture for developing service-oriented and component-based environmental models. Ecol. Model. 179, 61–77.
Paracchini, M., Pacini, C., Jones, L., Pérez-Soba, M., in press. An aggregation framework to link indicators associated with multifunctional land use to the stakeholder evaluation of policy options. Ecol. Indic., doi:10.1016/j.ecolind.2009.04.006. Corrected proof.
Parker, P., Letcher, R., Jakeman, A., Beck, M.B., Harris, G., Argent, R.M., Hare, M., Pahl-Wostl, C., Voinov, A., Janssen, M., Sullivan, P., Scoccimarro, M., Friend, A., Sonnenshein, M., Barker, D., Matejicek, L., Odulaja, D., Deadman, P., Lim, K., Larocque, G., Tarikhi, P., Fletcher, C., Put, A., Maxwell, T., Charles, A., Breeze, H., Nakatani, N., Mudgal, S., Naito, W., Osidele, O., Eriksson, I., Kautsky, U., Kautsky, E., Naeslund, B., Kumblad, L., Park, R., Maltagliati, S., Girardin, P., Rizzoli, A., Mauriello, D., Hoch, R., Pelletier, D., Reilly, J., Olafsdottir, R., Bin, S., 2002. Progress in integrated assessment and modelling. Environ. Model. Software 17, 209–217.
Pérez-Soba, M., Petit, S., Jones, L., Bertrand, N., Briquel, V., Omodei-Zorini, L., Contini, C., Helming, K., Farrington, J., Mossello, M.T., Wascher, D., Kienast, F., De Groot, R., 2008. Land use functions – a multifunctionality approach to assess the impact of land use changes on land use sustainability. In: Helming, K., Tabbush, P., Perez-Soba, M. (Eds.), Sustainability Impact Assessment of Land Use Changes. Springer, pp. 375–404.
Poppendieck, M., Poppendieck, T., 2006. Lean Software Development: An Agile Toolkit. Addison-Wesley Professional, 240 pp.
Potschin, M., Haines-Young, R., 2008. Making sustainability impact assessments: limits thresholds and the sustainability choice space. In: Helming, K., Tabbush, P., Perez-Soba, M. (Eds.), Sustainability Impact Assessment of Land Use Changes. Springer, pp. 425–450.
Raskin, J., 2000. Humane Interface: New Directions for Designing Interactive Systems. ACM Press, 256 pp.
Reynolds, J.F., Acock, B., 1997. Modularity and genericness in plant and ecosystem models. Ecol. Model. 94, 7–16.
Rizzoli, A., Young, W.J., 1997. Delivering environmental decision support systems: software tools and techniques. Environ. Model. Software 12, 237–249.
Rotmans, J., 1990. IMAGE an Integrated Model to Assess the Greenhouse Effect. Rijksuniversiteit Limburg, Maastricht.
Rotmans, J., Dowlatabadi, H., 1998. Integrated assessment modeling. In: Raynor, S., Malone, E. (Eds.), Human Choices & Climate Change. Batelle Press, Columbus, pp. 291–377.
Royce, W.W., 1970. Managing the development of large software systems. In: International Conference on Software Engineering TRW Monterey, California, United States, pp. 1–9.
Salo, O., Abrahamsson, P., 2008. Agile methods in European embedded software development organizations: a survey study of Extreme Programming and Scrum. IET Software 2, 58–64.
Schelhaas, M.J., Eggers, J., Lindner, M., Nabuurs, G.J., Pussinen, A., Päivinen, R., Schuck, A., Verkerk, P.J., Van de Werf, D.C., Zudin, S., 2007. Model Documentation for the European Forest Information Scenario Model (EFISCEN 3.1.3) Alterra, Wageningen.
Scholten, H., Kassahun, A., Refsgaard, J.C., Kargas, T., Gavardinas, C., Beulens, A.J.M., 2007. A methodology to support multidisciplinary model-based water management. Environ. Model. Software 22, 743–759.
Schwaber, K., Beedle, M., 2001. Agile Software Development with SCRUM. Prentice Hall, Upper Saddle River, 158 pp.
Sefelin, R., Tscheligi, M., Giller, V., 2003. Paper prototyping – what is it good for: a comparison of paper- and computer based low-fidelity prototyping. In: Conference on Human Factors in Computing Systems. ACM, New York, Ft. Lauderdale, Florida, USA.
Sieber, S., Müller, K., Verweij, P.J.F.M., Haraldsson, H., Fricke, K., Pacini, C., Tscherning, K., Helming, K., Jansson, T., 2008. Transfer into decision support: the sustainability impact assessment tool (SIAT). In: Helming, K., Tabbush, P., Perez-Soba, M. (Eds.), Sustainability Impact Assessment of Land Use Changes. Springer, pp. 107–128.
Simcoe, T., 2006. Open standards and intellectual property rights. In: Chesbrough, H., Vanhaverbeke, W., West, J. (Eds.), Open Innovation: Researching a New Paradigm. Oxford University Press, Oxford, pp. 161–183.
Snizek, B., Jorgenson, I., Acevedo, R., Biber, P., Seifert, S., Sieber, S., Jones, L., 2008. Tools for graphical representation of multidimensional data in SIAT. SENSOR Contract No. 003874 (GOCE) Deliverable number: 4.4.1 and 4.4.2, European Commission Within the Sixth Framework Programme (2002–2006), Brussels.
Srivastava, P.R., Kumar, K., 2009. An approach towards software quality assessment. Comm. Comput. Inf. Sci. 31, 150–160.
Standish Group, 2005. CHAOS Rising—A Chaos Executive Commentary. The Standish Group International, Inc., West Yarmouth, USA.
Sterk, B., Leeuwis, C., Van Ittersum, M.K., 2009. Land use models in complex societal problem solving: plug and play or networking? Environ. Model. Software 24, 165–172.
Sterk, B., Carberry, P., Leeuwis, C., Van Ittersum, M.K., Howden, M., Van Keulen, H., Rossing, W., 2008. The interface between land use systems research and policy: multiple arrangements and leverages. Land Use Policy 26, 434–442.
Tabbush, P., Frederiksen, P., Edwards, D., 2008. Impact assessments in the European Commission in relation to multifunctional land use. In: Helming, K., Tabbush, P., Perez-Soba, M. (Eds.), Sustainability Impact Assessment of Land Use Changes. Springer, pp. 35–54.
Tate, K., 2005. Sustainable Software Development, an Agile Perspective. Agile Software Development. Addison-Wesley, 226 pp.
Tractinsky, N., Katz, A.S., Ikar, D., 2000. What is beautiful is usable. Interact. Comput. 13, 127–145.
Van de Sluijs, J.P., 2002. Integrated assessment. In: Tolba, M.K. (Ed.), Responding to Global Environmental Change, Encyclopedia of Global Environmental Change. John Wiley & Sons, Chichester, pp. 250–253.
Van Ittersum, M.K., Brouwer, F., 2009. Integrated assessment of agricultural and environmental policies – concepts and tools. Environ. Sci. Policy 12, 543–545.
Van Ittersum, M.K., Ewert, F., Heckelei, T., Wery, J., Alkan Olsson, J., Andersen, E., Bezlepkina, I., Brouwer, F., Donatelli, M., Flichman, G., Olsson, L., Rizzoli, A., Van der Wal, T., Wien, J.J.F., Wolf, J., 2008. Integrated assessment of agricultural systems – a component-based framework for the European Union (SEAMLESS). Agric. Syst. 96, 150–165.
Verburg, P., Van Eck, J.R., De Nijs, T., Visser, H., Jong, K., 2004. A method to analyse neighbourhood characteristics of land use patterns. Comput. Environ. Urban 28, 667–690.
Verweij, P., Knapen, M., Wien, J., 2007. The use of OpenMI in model based integrated assessments. In: MODSIM 2007 International Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand Christchurch, New Zealand.
Verweij, P., Sieber, S., Wien, J., Müller, K., 2006. SIAT, a sustainable impact assessment tool for understanding the drivers in integrated impact assessment. In: International Congress on Environmental Modelling and Software iEMSs 2006, Burlington (Vermont), USA.
Verweij, P.J.F.M., Roller, J.A.T., Vanmeulebrouk, B., Knapen, M.J.R., Van Randen, Y., De Winter, W.P., Franke, G.J., 2009. SIAT – system architecture and design. SENSOR

Contract No 003874 (GOCE) Deliverable number: 4.3.1, European Commission within the Sixth Framework Programme (2002–2006), Brussels.
Villa, F., Athanasiadis, I., Rizzoli, A., 2009. Modelling with knowledge: a review of emerging semantic approaches to environmental modelling. Environ. Model. Software 24, 577–587.
Voinov, A., Gaddis, E.J.B., 2008. Lessons for successful participatory watershed modeling: a perspective from modeling practitioners. Ecol. Model. 216, 197–207.
Westhoek, H., Van den Berg, M., Bakkes, J.A., 2006. Scenario development to explore the future of Europe's rural areas. Agric. Ecosyst. Environ. 114, 7–20.
Wien, J.J.F., Rizzoli, A., Knapen, M.J.R., Athanasiadis, I., Janssen, S., Ruinelli, L., Villa, F., Svensson, M., Wallman, P., Jonsson, B., Van Ittersum, M.K., 2010. A web-based software system for model integration in agro-environmental impact assessments. In: F. Brouwer, M.K. Van Ittersum (Eds.), Environmental and Agricultural Modelling: Integrated Approaches for Policy Impact Assessment. Springer Academic Publishing, in press.