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).