The Use of UML for Model Design and Scientific Software Development

C. R. Maul
Institute of Sustainable Irrigated Agriculture, (ISIA), Tatura VIC 3616, Australia
Christian.Maul@nri.vic.gov.au

Key words: UML, Software Engineering, Model Design

Abstract: UML is a visual modelling language and can provide a clear description of the structure and the behaviour of a model. Structure is portrayed by component and class diagrams. Behaviour is detailed by use cases, diagrams of sequence and collaboration, state charts, and activity diagrams. To benefit most from UML, the software development process should be use case driven, architecture focussed, iterative and incremental. The new standard of UML provides a complete set of tools to describe every aspect of a model. UML has proven a valuable tool in software projects [7,8]. It can communicate software requirements not only to those within a software project but also to stakeholders and the interested public. It can support the model development process, speed up the software development process considerably, and assure the implementation of robust, reliable software that is easy to alter.

1. INTRODUCTION

Scientific models and their software implementation typically lack architecture, and it is also difficult to alter the model and subsequently the software in order to incorporate scientific developments. The software development process is rarely understood as an ongoing, iterative and incremental one. Models usually implement what happens to be the current state of art. Lack of continuity in funding and personnel often impedes further development or the realisation of a strategic view. As a consequence every project starts from scratch and long-term projects have difficulties being funded at all.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35503-0_29

R. Denzer et al. (eds.), *Environmental Software Systems*

The software industry has developed many procedures to assure the quality of software products and to assure continuity in the release of versions that extend functionality. The most important is the object-oriented approach. Merits, advantages and problems of the object-oriented design will not be repeated in this paper ((Maul et al. 1998). Unified Modelling Language (UML) has been developed as an additional technique. It enables the realisation of the advantages of object-oriented analysis and design. UML is firmly connected to the object-oriented development process. It is, however, a modelling language that is not connected to any particular object-oriented language for the implementation. Smalltalk, Lisp, Eiffel, Oberon, C++, Java or Delphi can all be used with UML. UML can also adapt to different software development processes. This paper will show how to realise the potential advantages of object-oriented analysis and design for environmental software by using UML.

Software projects in science and business differ in their size, complexity, division of labour and impact on the organisation. Nevertheless, science and engineering departments can select appropriate methods from those developed for business. The following scheme describes the model building process:

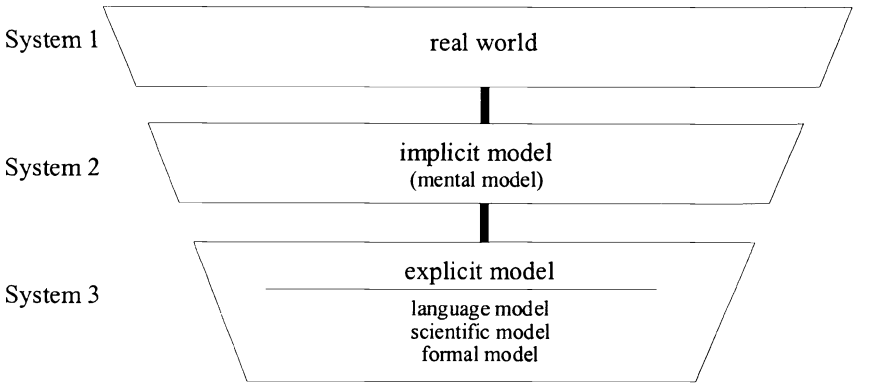

Figure 1. The model building process

The model building process comprises two major phases (System 2 and 3). The first phase results in an implicit or mental model, which then forms the basis for an explicit model developed in the second phase. The form of trials and data collection is determined by any assumptions in the implicit

model. A modelling language that is close to natural language is desirable during the first phase. This means the model structure is more easily captured. As soon as measured data are available a model is constructed that summarises knowledge and tries to gain insight into the system structure and behaviour (Lentz 1998). The description shifts from qualitative to quantitative, from verbal description to formulae, numbers, data tables and assessments. A useful modelling language is one that can be used during all three stages of the explicit model, preventing information being lost by translation from different notations that represent different types of knowledge. The language must be understood intuitively to allow broad discussion of the model and improvements. Discourse in the scientific community is the only way to push forward the key abstractions that finally form the model. The expertise of scientists who do not care about modelling and software implementation must be incorporated. The goal must be to communicate and to discuss the modelling problems as scientific issues and not as data processing problems. Similar communication problems and misunderstandings in business have been one important reason for the emergence of UML.

The standardisation of different methods into UML (Booch et al. 1999) has changed business considerably. However, scientists have not taken up these developments. Clearly, UML has an enterprise information systems background. Nevertheless, it provides the vocabulary and rules to create and read any well-formed model. It does not dictate which model to create (Booch 1998). Thus, its application to scientific models and the scientific software development process is feasible and might be beneficial.

2. METHOD

UML is basically a language used to describe software systems. For the sake of universal applicability no particular software development process is prescribed. This has been done to integrate the different predefined development processes of companies such as IBM, Microsoft, Oracle, Siemens, KPMG and Anderson. Apart from the expectation of wider application of UML it is impossible to define one valid software development process that suits every corporate culture. Logically, UML offers only a framework and notation for processes. This makes it applicable to the scientific development process, which is different from an industrial one. However, as Wahl [10] correctly emphasises, any software development without a defined process is unprofessional. For a possible software development cycle description, see Booch et al (1999).

UML must be understood as part of the endeavour to structure the understanding of a particular system and to structure the software development process instead of mindlessly amassing source code. The most benefit from UML will be derived from a process that is:

- iterative and incremental,
- use case driven, and
- focussed on architecture (Booch et al. 1999)

3. DISCUSSION AND CONCLUSION

An important problem in software development is the lack of structural connection between the logical model and its implementation. This usually causes problems in the later stages of the project. UML highlights the common features and differences between views by describing different views with the essentially the same diagrams. This facilitates decisions about how changes in the model must precipitate in the source code. Particularly the CASE tool (Computer Aided Software Engineering) Rational Rose™ in particular realises the different views in an exemplary manner. The better these views are separated and documented, the easier it is to alter and maintain software and to restart a development project.

UML is a design tool and offers scaffolding for the model development process. It is necessary to keep in mind that the primary product of a development team is not beautiful documents, world-class meetings, great slogans or Pulitzer Prize-winning lines of source code. Instead, it is good, robust software that satisfies the needs of its users, the scientific community or a business. Everything else is secondary (Booch 1999). However, secondary does not mean irrelevant.

UML, with its variety of diagrams, connections and definitions, requires some effort to be understood fully. To learn UML in detail needs time. The reward is that it can speed up the development process significantly when used in conjunction with CASE tools such as Rational Rose, TogetherJ or SimplyObjects. In scientific models, however, there are usually problems in the beginning, because the capture of functional requirements of the software tends to be fuzzy. If there is a client, he does not usually know exactly what he wants. Thus, frequent iterations are required to refine functional requirements. Furthermore, it is unfortunately in the interest of the applicant of a research grant to be as vague as possible to assure that the project will be a success. This turns against developers when it comes to implementing software. The use case driven approach enables scientists to focus on what the software should be able to do and to clear up problems at the start.

Once clients know what they want, they want it immediately. UML tools allow rapid prototyping to develop a simple model that can be refined later. Because a system may not be very well understood, this can lead to the problem of too many changes. Alterations usually lead to the deterioration in software architecture and source code. UML always provides a clear starting point for code generation, particularly in the later phases if the model does not work realistically and new parts must be integrated and designed. UML shows its strength because it is designed to be part of an iterative process.

It is definitely an advantage of UML that it is not connected to a particular software development process. This makes it suitable for scientific development processes where most work is done during the analysis and design phase. It is conceivable to use commercial services for the implementation of the model or for parts of it such as the graphical user interface. UML can serve as a tool to connect the different parts and to bring different development teams together. Nearly all CASE tools that use UML implement development teamwork.

Because there are many proper ways to develop a model, the model building process is also labelled an art rather than a science (Booch 1999). UML converts this art into a craft. The class diagram can follow natural language in the naming of objects, variables and their connections and also offers the opportunity for implementation in different program languages. The various behavioural diagrams assure a well-defined, coherent and intelligible model.

The scientific software development process is characterised by:

- small teams,
- frequently adapted components or objects,
- disruptions in funding, personnel and changing size of project groups,
- divergent interests of team members (for example, if parts are realised as a thesis),
- changing focuses in development, and
- a changing model base through further scientific development.

All these characteristics require intensive communication within the development team and to affiliated scientists. It also requires sufficient and good documentation. This creates a challenge for project management that can be solved by using appropriate and compatible tools in the model and software development process.

The effort in the analysis and design phase is clearly higher than it is with procedural programming. This will be repaid later, however, when the architecture of the software is clearly recognizable, when the scope of

variables and methods is restricted, and when changes in the program do not develop into a nightmare.

UML has proven a valuable tool in software projects realised so far [7,8]. It can support the model and the software development process tremendously. It makes software development easier to plan and calculate.

UML serves in a project in many ways:

- analysing and designing any model,
- communicating a design within the team or to stakeholders,
- communicating the model or parts to experts in natural language,
- portraying models in scientific papers,
- providing robust architecture that is easy to alter,
- providing opportunities to incorporate future scientific developments,
- specifying parts, if the system is developed in co-operation,
- defining software components,
- serving as a starting point for program documentation, and
- using CASE-tools and generating code.

4. REFERENCES

BOOCH, G (1998): The Future of Software. Seminar given in Melbourne 20.11.98.

BOOCH, G, RUMBAUGH, J, JACOBSON I, (1999): Unified modeling Language. User Guide. Addison-Wesley, Menlo Park, CA, US.

LENTZ, W (1998): Model applications in horticulture: a review. Scientia Horticulturae 74:151-174.

MAUL, CR; GOODWIN, I; KOCH, B (1998): Object Oriented Software Engineering for Agricultural Models. AgEng'98 International Conference on Agricultural Engineering 24.8. - 26.8.1998. Paper No 98-A-068, Oslo, No.