From Components to Decisions: The Role of Software Engineering and Frameworks in Catchment Decision Support

R. M. Argent[a], R. A. Vertessy[b], J. Rahman[b], S. Seaton[b]

[a] *Centre for Environmental Applied Hydrology and Cooperative Research Centre for Catchment Hydrology, Department of Civil and Environmental Engineering, The University of Melbourne, 3010, Australia. (R.Argent@unimelb.edu.au)*

[b] *Cooperative Research Centre for Catchment Hydrology, CSIRO Land and Water, Canberra*

Abstract: Good simulation modelling demands the tailoring of models to problems. Such tailoring is greatly aided by the re-use of model components (modules), which, in turn, is supported by modern software engineering practices. While scientists working in catchment modelling have been early adopters and continual users of computer-based models, many models have been, and continue to be, developed without taking advantage of modern software engineering techniques. Adoption of good modelling practices can reduce module development and debugging time, simplify model construction and maintenance, improve communication of model details, facilitate simulation result comparisons and analysis, and greatly aid many aspects of modelling for decision support. Improvements in decision support arise not only from higher quality, more flexible and more robust tailored products, but also from the increased time made available for involvement of, and communication with, decision makers. Modelling frameworks form a keystone in adoption of good modelling practice, by providing an enabling environment within which developers have access to many tools that support the modelling task. Core simulation modelling components are supported in a framework by suites of service modules, such as data handling and generation, output management, comparison, analysis and visualisation tools. Other benefits of frameworks include a common software operational paradigm and consistent delivery approach to decision makers. This paper showcases many of the software engineering techniques that are relevant to the component-based modelling approaches being adopted in ecological and bio-physical simulation modelling, and which have been included in a catchment prediction modelling framework being developed by the Cooperative Research Centre for Catchment Hydrology. This framework forms a basis for integration of existing models and newly developed models into a catchment prediction toolkit that aims to service some of the decision support modelling requirements of the catchment management industry.

Keywords: Component-based modelling; Modelling frameworks; Good modelling practice

1. INTRODUCTION

While hydrologists have been early adopters and continual users of computer based models, many models have been developed prior to widespread use of proven software engineering principles. Even today, new developments often fail to take advantage of the improvements to software quality available through the adoption of modern software engineering techniques. With an increasing focus on integrated environmental modelling, the use of sound software engineering practices becomes even more important.

More than a decade ago, Laut and Taplin [1989] remarked:

> *Perhaps the most serious problem in model development for catchment management in the near future is that of integrating present-day single purpose models to encompass the variety of land resource types and land uses in a medium to large catchment, with the purpose of defining the consequences of land use change on stream flow and water quality. [p.22]*

This remark has been echoed over the years as integrated environmental management has developed, and the use of modelling to support environmental decision making has expanded [Argent et al., 1999; Born and Sonzogni, 1995; Cairns Jr. and Crawford, 1991; Margerum and Born, 1995; Mitchell and Hollick, 1993].

Three issues that are often raised in the area of decision making using integrated environmental modelling tools are those of conceptual incommensurability, scaling of temporal and spatial model representations, and software support. One of the aims of the research reported here is to improve model development practices by encouraging and supporting the use of sound software engineering principles. Adoption of these practices will allow modellers to quickly develop models that are both powerful and flexible as well as being easy to communicate and maintain. In the long run, this will result in improved support of decision making through removal of modelling sticking points, improved communication of models and provision of more flexible modelling applications.

One of the challenges to changing practice in environmental modelling is to sort out those practices that were developed in computer science, and adopted in business enterprise development, that are actually useful, practical and relevant. The following sections cover some approaches and techniques that demand consideration.

2. TECHNIQUES FOR IMPROVED MODEL DEVELOPMENT PRACTICE

The techniques that follow can improve the model development process and the quality of modelling applications as a whole. Each technique is largely independent, allowing developers to adopt techniques over time. The techniques include process improvements, software engineering principles, and specific tools.

2.1 Communication of Models

When we write models, we expect to be communicating details of the model to various groups. We communicate the scientific details and assumptions to our peers and clients, data requirements to our users, and software structure to programmers and researchers.

These communications often take the form of written documentation, such as journal articles, manuals, emails, code comments and software structure diagrams. These documents are helpful at different times in a modelling project and focus on different groups of people.

A major problem with documentation is the time taken to produce it before, during or after model development. This discourages the production and maintenance of documentation, which in turn leads to inconsistency between model and documentation. Even when documentation is present, it usually isn't available in different forms to suit the different groups. For example, a programmer employed to make modifications to a model, would benefit greatly from written and diagrammatic documentation of the software structure, but may have to rely solely on journal articles that describe the scientific approach of the model.

Thus, improved documentation of models, with the various uses in mind, should be done. At a minimum, algorithms should be listed and described, and user manuals and context sensitive help must be used.

Improved model communication requires easier methods for creating and maintaining model documentation. Automated documentation tools, such as JavaDOC and DOC++, utilise the source code of a model, allowing the quick creation and modification of documents. There is a range of automatic documentation tools available, largely supporting object and class based modelling approaches. By using automatic tools we can ensure consistency between model and documentation, match standards dictated by an organisation, and always have documentation available.

2.2 Metadata

An extension of good model documentation is the appropriate use of metadata. Adopting a metadata strategy for all modelling within an organisation is an approach that can pay considerable dividends. By developing models that insist on appropriate metadata, and incorporating metadata into modelling results, a modelling history is created.

Metadata describes properties of the data (such as its name, author or license) that places the data in a context and assists in its interpretation. The attributes that constitute "useful metadata" vary between industries and even between projects. Commonly used fields for environmental metadata include location, quality, size, and how the data were created and collected.

Metadata is an area where something is better than nothing. While the amount of information, and the way information is structured, varies between metadata schemes, consistency is the key. Consistent metadata improves the quality of data assessments and reduces interpretation time. More information supports powerful data mining

but becomes a hindrance when the information is not available consistently across a data resource.

A consistent style will involve creating metadata that has the same essential set of attributes for each dataset. This set of attributes may be as simple as:

- Dataset Name:
- Variable Description:
- Units:
- Creation Date:
- File Name:
- Location of File:
- Comments:

As long as the base set of attributes is consistent, the metadata will remain usable. Current metadata standards and guidelines, such as the Content Standard for Digital Geospatial Metadata (CSDGM) and the Australian and New Zealand Land and Information Council (ANZLIC) Guidelines on Core Metadata Elements, provide useful examples. Adopting a simple scheme can be the first step towards effective management of data resources and providing a history of modelling efforts.

2.3 Use and Abuse of Constants

One problem in re-using software is associated with "hard wiring" of constants in models. By carefully considering the nature of all constants used in a model, and managing constants appropriately, it is possible to vastly increase the flexibility and re-usability of software.

If you expect your modelling to be successful, expect it to be reused. If you expect it to be reused, then efforts should be made to make it flexible. Models tend to be developed to suit a particular modelling exercise, which often involves constants including time-steps, grid spacings and geographic locations. Successful modelling exercises often lead to models being applied in different situations, where the same constants may not apply. Adapting a model to a new situation, where the previous set of constants is embedded in the code, can be time consuming and error prone. The modifications create a fork in the model development, where the two versions of the model proceed in different and possibly incompatible directions. If a bug is fixed in one version of the source code, it is unlikely to be fixed in the other, unless sound version control and software management is in place. The model's code may be written based on assumptions about the acceptable values of the constant parameters. When these assumptions are made, but not well documented, they make applying the model in new situations difficult.

A good starting point is to think of everything as a parameter and, therefore, subject to change. This can include various artefacts of the software process, such as loop increments. For anything that is considered a constant, decide whether it should be categorised as a true constant, a project constant or a simple constant. True constants will never change, so that a value can be used for every conceivable location on Earth, at any reasonable time in history, for any valid scenario. When you have a true constant, define it as a constant within the source code, and use the constant name in subsequent code.

Project constants can be thought of as constants that are applicable for most of the current likely applications of the software, while simple constants are those that are likely to remain constant for a given application run.

Initialisation (INI) files store parameters and options for programs that need the flexibility to change, but are unlikely to change on a regular basis. Thus, project constants should be stored in INI files, while simple constants should be used in parameter files that are accessed for particular runs.

3. COMPONENT BASED MODELLING

Component Based Software Engineering (CBSE) is an approach to software construction that emphasises viewing a system as a collection of interchangeable, and largely independent, components. The components can be designed specifically for an application, reused from a previous effort, or purchased from a component vendor. The advantages of keeping a model's core scientific algorithms separate from the support code include improved maintainability, easier communication and an easier path to integration in modelling frameworks.

CBSE is typically supported by a component standard, such as Microsoft's Component Object Model (COM) or Sun Microsystems's Enterprise Java Beans (EJB), that defines certain minimum standards with which components should comply. These minimum standards allow the component standard to provide infrastructure for managing components, such as creating instances of components and making components available over the network.

3.1 Environmental Modelling with Components

There are numerous areas where CBSE can be applied in environmental modelling efforts. Initially, commercial off-the-shelf components can be used to handle visualisation and user interface

tasks within a modelling application. This approach has been taken, for example, in the Model for Urban Stormwater Improvement Conceptualisation (MUSIC), developed by the Cooperative Research Centre for Catchment Hydrology, where a graphing component is used to display time series data and a diagramming component is used to interactively design a stormwater treatment system. The next step is to implement a model's core scientific code as a component, or collection of components that can be supported by a modelling framework that provides model integration capabilities.

3.2 Using and Combining Components

Components are typically used as objects in an object-oriented programming language, such as C++. Components, like objects, support an interface that contains operations that can be performed, queries that can be answered and properties that can be changed. Client code is written to make use of these interfaces to perform required component functions.

Visual components, such as charts and buttons, are typically arranged on a window using a user interface design tool, such as Visual Basic or Borland Delphi. Code can then be added to the window to control the behaviour of the component, such as providing the chart with data or specifying an action to be performed when the user clicks on the button.

3.3 Interchanging Components

When a component needs to be exchanged with another, such as switching between two different rainfall-runoff models, there are a number of possible approaches in a component based system. Many component based systems support interchanging components that share a common interface. This approach is similar to the object oriented notion of inheritance, with the common interface being synonymous with a common parent class.

Interactive component based modelling systems, such as the Integrated Catchment Modelling System (ICMS), allow components to be interchanged using a graphical display [Reed et al. 1999]. Here the two components may not share the same formal interface, but still have similar enough elements that a human operator can resolve the differences.

3.4 Design and Use of Components

In designing components, there are techniques available that improve the ability to detect and fix programming bugs. These approaches, such as Design By Contract [Hunt and Thomas, 2000], set up the specification for the interaction of variables within components, and ensure that if an incorrect variable type is passed to or from a routine, that the error is picked up, and the source identified.

Collectively, the techniques described in the preceding sections can be used to improve the quality of environmental modelling software. This, in turn, can lead to improvements in the scientific process of model development. First, and foremost, by reducing the time it takes to maintain and debug models, we free more time for developing the science. Improving the communication of model details makes it easier to share developments with peers and have them advance the work. Finally, improved maintenance and communication collectively lead to lower defect rates in the software, giving a better end product and reduced software costs.

Software design techniques that recognise the need for the software to remain adaptable allow the model to change and grow over its lifetime. We suggest the use of modelling frameworks to improve productivity by relying on a framework's built in data management and input-output routines.

4. MODELLING FRAMEWORKS

While component standards, such as COM or EJB, and programming languages, such as Java, provide support for creating and connecting software components, they provide little other infrastructure. Frameworks can be built with, and on top of, a component standard or programming language in order to provide additional services and infrastructure useful to a particular class of applications. Frameworks typically impose additional constraints on the way in which components can interact, in order to provide high-level management functions to developers. Environmental modelling frameworks typically include infrastructure for managing data and the use of data within the system, models and the integration of models and visualisation of model results. This functionality can be written once, by the framework developers, and used repeatedly by model developers.

Typically, the more support and infrastructure offered by a framework, the narrower the class of applications that can be adequately supported by the framework. For example, an application framework may contain user interface concepts, such as buttons and entry fields useful to any graphical application. This functionality can be

used to support the development of modelling applications, web browsers or word processors.

An important consideration in selecting or developing a modelling framework is to ensure that the framework provides a strong level of support for your class of applications, without imposing too much in terms of developmental overhead or model operation. If a framework imposes a considerable overhead, then developers used to working in a particular way will not adopt it. A straight-jacket framework unduly dictates the way models operate, reducing the opportunity for developers to implement elegant software solutions.

Alongside issues such as consistency and component re-use, the adoption of framework-based component modelling also has long term software maintenance advantages. Maintainable software will have a longer useful life with a lower total cost of ownership. The software engineering industry has developed techniques for designing and structuring software to improve maintainability. Techniques for improving communication of software details also work to improve the maintainability as it reduces the effort required to understand and modify the software. Modelling frameworks encourage modularity, by providing reusable modules for model and data management, which in turn leads to more maintainable structures. Careful attention to the way constants are handled, within the modelling application, improves the adaptability of the model to new projects.

Modelling frameworks form a keystone in adoption of good modelling practice, by providing an enabling environment within which developers have access to many tools that support the modelling task. Team-based model building can also be supported. Core simulation modelling components are supported in a framework by suites of service modules, such as data handling and generation, output management, comparison, model calibration, analysis and visualisation tools. Other benefits of frameworks include a common software operational paradigm and consistent delivery approach to decision makers.

4.1 Support for Decision Making

Adoption of CBSE principles and practices, particularly with the use of modelling frameworks, also benefits environmental decision making. Three aspects of decision support include:

- Reduced time spent on model development, hence reduced cost and more timely delivery of decision support tools;
- Improved communication of model structure and assumptions, allowing a better fit between the model and the problem being addressed, and
- Increased flexibility in software operation and application, allowing models to be applied to a wider range of problems.

Beyond these advantages, the use of CBSE and modelling frameworks also supports the user based development paradigms that are a part of modern software development approaches. These include prototyping approaches that allow users to interact with an application, and become familiar with data formats and information presentation, long before the model algorithms are finalised.

5. CONCLUSIONS

The practices discussed in this paper have been identified as those from software engineering and business application sciences that provide considerable promise to assist the further development of integrated environmental modelling and decision support tools. However, software development practice is only one of the key issues of integrated environmental modelling. Other issues to be addressed as we go forward lie in the areas of 'disciplinary stiffness' and 'scale mismatches'.

Disciplinary stiffness stems from the relatively narrow knowledge base and modus-operandi of each disciplinary group participating in catchment prediction. There is a huge gulf in the language and analytical tools used by social scientists, economists, ecologists and hydrologists, so it is not surprising that they rarely attempt to couple their world views. Modellers are unlikely to work with models from outside their fields unless the most exacting standards of good modelling practice have been adhered to in the development of those models. As discussed earlier, these would include rich documentation, proper use of constants and clear model structure. To complicate matters further, a rich diversity of modelling methods is employed across these fields, ranging from conceptual and statistical, through to probabilistic and deterministic approaches based on the solution of partial differential equations. Any framework seeking to support integrated modelling across disciplines must cater for this diversity of methods.

Scale mismatches are another area for continued investigation. Catchment managers need models that can span space scales from points and paddocks (10^2-10^3 m^2) to whole regions (10^6 km^2) and even up to continental scales. In the time

dimension, managers seek models that can predict catchment function on scales ranging from minutes to centuries. This broad space and time domain cannot be spanned by any single model. Instead, discrete models populate this space, occasionally overlapping in their range. Notwithstanding the difficulties of coping with this range, problems arise because the spatial and temporal units that hydrologists specify in their models may differ markedly from those employed by ecologists or economists. Even within the hydrology domain, coping with the spatial and temporal scale mismatches of different catchment processes has been a major impediment to integration.

If we take for example the integration of sediment and nutrient generation models with salinisation models, one is confronted by the need to have two very different levels of spatial and temporal resolution. Most sediment and nutrient generation and movement takes place during storm events, and the vast bulk of this occurs during short bursts of high intensity rainfall. Furthermore, the fate of mobilised material depends strongly on the topography, degree of connectedness between hillslopes and streams and the nature of the land cover, particularly in the riparian zone. Models of this process thus require comparatively fine temporal and spatial resolution. On the other hand, salinisation is a very slow process that takes place over decades or centuries. Whilst topography is also an important variable, salinisation models can cope with much coarser topography information than models of sediment and nutrient movement.

The 'brute-force' solution to the scale mismatch problem is to have integrated models operate on the minimum temporal and spatial scale. Such an approach is highly inefficient and unparsimonious. A more sensible way forward is to provide a set of scaling procedures that can be used to adapt particular space and time elements in models for various processes. Any framework designed to support integrated modelling should contain within it, scaling procedures to aggregate and disaggregate spatial and temporal data.

By working on the issues discussed here, and adopting some of the elements of good software engineering in our modelling practice we can continue to develop our ability to support well-informed environmental management.

6. ACKNOWLEDGEMENTS

This research is supported by the Cooperative Research Centre for Catchment Hydrology.

7. REFERENCES

Argent, R.M., R.B. Grayson and S.A. Ewing, Integrated models for environmental management: Issues of process and design. *Environment International*, 25(6/7): 693-699, 1999.

Born, S.M. and W.C. Sonzogni, Integrated environmental management: strengthening the conceptualization. *Environmental Management*, 19(2): 167-181, 1995.

Cairns Jr., J. and T.V. Crawford, *Integrated Environmental Management*, Lewis, Chelsea, 1991.

Hunt, A. and D. Thomas, *The Pragmatic Programmer: From Journeyman to Master*, Addison Wesley, 352 pp., 2000.

Laut, P. and B.J. Taplin, Catchment Management in Australia in the 1980s. CSIRO Division of Water Resources, Divisional Report 89/3, 252pp., 1989.

Margerum, R.D. and S.M. Born, Integrated environmental management: moving from theory to practice. *Journal of Environmental Planning and Management*, 38(3): 371-391, 1995.

Mitchell, B. and M. Hollick, Integrated catchment management in Western Australia: Transition from concept to implementation. *Environmental Management*, 17(6): 735-743. 1993.

Reed, M., S.M. Cuddy, and A.E. Rizzoli, A framework for modelling multiple resource management issues - an open modelling approach. *Environmental Modelling and Software*, 14, 503-509, 1999.