Software Development Best Practices in Integrated Environmental Model Development

Takuya Iwanaga  
*Australian National University*, iwanaga.takuya@anu.edu.au

Joel Rahman  
*Flow Matters*, joel@flowmatters.com.au

Daniel Partington  
*Flinders University*, daniel.partington@flinders.edu.au

Barry Croke  
*Australian National University*, barry.croke@anu.edu.au

Anthony J. Jakeman  
*Australian National University*, tony.jakeman@anu.edu.au

Follow this and additional works at: [https://scholarsarchive.byu.edu/iemssconference](https://scholarsarchive.byu.edu/iemssconference)

[https://scholarsarchive.byu.edu/iemssconference/2018/Stream-A/18](https://scholarsarchive.byu.edu/iemssconference/2018/Stream-A/18)

This Oral Presentation (in session) is brought to you for free and open access by the Civil and Environmental Engineering at BYU ScholarsArchive. It has been accepted for inclusion in International Congress on Environmental Modelling and Software by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
Software Development Practices in Integrated Environmental Model Development

Takuya Iwanaga¹, Joel Rahman², Daniel Partington³, Barry Croke¹,⁴, Anthony J. Jakeman¹
¹Fenner School of Environment and Society, Australian National University, Canberra, Australia
iwanaga.takuya@anu.edu.au
²Flow Matters, Canberra, Australia
³National Centre for Groundwater Research and Training, Flinders University of South Australia, Adelaide, Australia
⁴Mathematical Sciences Institute, Australian National University, Canberra, Australia

Abstract: Integrated models are often made up of smaller component models, each representing a particular domain that are coupled together. Such models are software for a scientific purpose, and so similarities between model and software development exist. These models tend to be developed by researchers who take on a dual-role of scientist and software developer. Despite the similarities in development approaches, many best practices found within the field of software engineering may not be applied to the development of integrated environmental models. The absence of best practices lead to issues revolving around the reusability, interoperability, and reliability (in terms of model application and results) of models developed for purposes such as integrated assessment. To address these concerns recent efforts have seen the development and proliferation of component-based implementation approaches and model development frameworks. These approaches by themselves are not a panacea to model development issues. Component models may be difficult to integrate and error-prone due to the lack of best practices used in their construction. Also, the structure of component and integrated models may render them difficult to reuse/reapply in a different context. Model development frameworks may ease the overall technical burden of model development and integration, however they often come with a steep learning curve of their own, which may hamper their effective use. This in turn exacerbates issues regarding model reusability and interoperability.

In this paper, we introduce software development practices identified through literature review and expert knowledge such as code reviews, testing, and code management. Such practices may be useful to model developers in the development of reusable and reproducible integrated environmental models. These practices are compatible and complementary to existing development practices, such as iterative component-based approaches, and serve to fill in the gap between software and modelling paradigms.

Keywords: integrated environmental modelling; environmental model development; model development practices; software development practices

1 INTRODUCTION

Integrated modelling is a multi/trans-disciplinary approach to model development that encompasses knowledge and information from diverse domains of knowledge to represent a system of systems. An integrated approach to modelling is often necessary to assess environmental issues as they cannot be neatly contained within the lines of traditional academic disciplines. A holistic systems approach is therefore necessary when assessing complex inter-connected systems that traverse the socioeconomic, and biophysical domains.
Integrated Environmental Models (IEMs) are applied across a wide range of contexts from decision support, policy analysis and, more recently, as a tool to explore possible outcomes – the effects and impacts – that arise from a suite of policy and management options. One reason for doing so is to inform discourse on environmental issues (De Vos et al., 2011; McIntosh et al., 2011), which may be time-sensitive ‘wicked problems’. These are problems that are difficult to resolve and continue to increase in difficulty as time goes by.

Arnold (2016) describes a process for managing such environmental issues as having three embedded iterative cycles. These cycles roughly correspond to the model development and modelling process (technical cycle) which serves to inform and increase knowledge of the modelled system (conceptual learning cycle) which in turn feeds into the development of appropriate policy measures and management objectives (policy cycle). Timely resolution of environmental issues may depend on the speed at which we iterate through these cycles. Maximizing the effectiveness and efficiency of the IEM development process is one measure that may ‘speed up’ such a process.

The development of IEMs, however, requires significant time and resources, and a wide variety of specialist domain knowledge. This is compounded by the increasing complexity of IEMs as model developers seek to expand the number of system domains represented to capture second order (and higher order) interactions which were previously not considered. The increasing complexity is illustrated by the historical publication trend and diversity of keywords that appear in the literature over time (e.g. Zare et al. 2017). Domain knowledge refers not only to each social and biophysical system represented in the IEM but extends to the technical software development (SD) skills required to develop and integrate these models.

Hamilton et al. (2015) describe recent advances in integrated assessment as having pushed the field into a state of ‘maturity’. The time then is ripe for further introspection and reflexive analysis on the methods and approaches used in IEM development. In this paper we explore common approaches to environmental model development and discuss SD practices that may be helpful, with the aim of increasing the speed of iteration through the management cycle. As an aside, readers with a background in software engineering are advised that terminology that is highly specific to software development is purposefully avoided in this paper.

Counterparts to commonly applied approaches in software engineering can be found in IEM development. These include the use of component-based, iterative and participatory approaches. Common SD practices identified here include code review, testing, and management. Adopting these practices may further IEM development practices and alleviate current issues with model reusability and reproducibility.

2 MODEL DEVELOPMENT FRAMEWORKS

Environmental models are considered important tools that aid in managing complex interconnected systems. Their development requires significant resources and domain specific knowledge. For a variety of reasons IEM developers often work in a dual-capacity; as both researcher and software developer. As much as 30% of researchers’ time is spent developing software across the sciences (based on 1,972 respondents, Hannay et al. 2009). Although the respondents were not specifically environmental modelers the reported figure is used here to indicate the significant amount of time generally spent developing scientific software by those without formal SD training.

A common approach to IEM development is the coupling of multiple single domain models. Model Development Frameworks (MDFs) provide a common computational platform through which these models are integrated. MDFs can be categorized as being large scale – in terms of budget, scope, or ubiquity – and smaller scale frameworks that are purpose built. Such frameworks aim to resolve or alleviate commonly encountered model development tasks and issues. These include visualization, data analysis and format conversion, and uncertainty and sensitivity assessment (Prabhu et al., 2011; Wirtz and Nowak, 2017). Such included capabilities reduce the duplication of work across individual researchers, teams, and projects and as a result large gains in productivity can be expected once the initial learning curve is overcome. Examples of MDFs as defined here include eWater Source (Carr and Podger, 2012), APSIM (Holzworth et al., 2014), and ArcGIS (ESRI, 2011).
While large scale MDFs may provide tooling to address common model development concerns, research is often conducted in highly specialized contexts and so a given MDF may be unsuitable outside such contexts. Additionally, whilst MDFs are intended to ease the technical complexity of model development and integration they, ironically, are accompanied by a steep learning curve of their own. Use of an MDF may then be inappropriate in cases where such complexity is unwarranted for the model purpose. It is noted that even if a general universal MDF were to be developed, some amount of software development would still be necessary (Ahalt et al., 2014). Smaller scale frameworks are often built using open source programming languages and the numerous packages available for those (e.g. Kneis, 2015; Malard et al., 2017). In such cases IEM developers may construct specific custom-made frameworks and interfaces, or avoided the use of interfaces altogether at the cost of reusability (Belete et al., 2017). In this manner, the potentially high adoption cost is shifted to a potentially lower cost bespoke integration solution that is purpose-built.

MDFs, regardless of their scale and type, typically rely on interfaces to achieve model integration. Interfaces can be thought of as software intermediaries through which data passes between the framework and the component model, handling the data exchange process as necessary (Peckham et al., 2013). To achieve this, interfaces require metadata detailing the semantic information that defines the expected inputs and outputs, which constitutes a hard requirement to achieving interoperability between models. Metadata is provided according to an interface specification which may require additional code to be written or annotated with semantic information (David et al., 2013; Lloyd et al., 2011).

There is, however, no unified interface specification in the environmental modelling community resulting in a variety of specifications that have been adopted and applied to some degree (Elag and Goodall, 2013; Parmar et al., 2016). The Open Modelling Interface (known as OpenMI, Castronova et al., 2013) is one such example that has been adopted and applied across a variety of MDFs and modelling contexts (Betrie et al., 2011; Knapen et al., 2013; van Ittersum et al., 2008). Currently available interface specifications are reported to be too biased towards one system domain or another, reflecting their original domain-specific purpose. The original focus of OpenMI, for example, was on surface and groundwater hydraulics, since expanded to better accommodate other system domains (Donchyts et al., 2010). Some interfaces require component models to be modified to enable their use from within an MDF (Lloyd et al., 2011), which may be undesirable if the aim is to develop components with (near) seamless reusability. On a positive note, the need for a unified specification is recognized and continues to be addressed (Donchyts et al., 2010; Elag and Goodall, 2013; Laniak et al., 2013).

2.2 Iterative Participatory Development

Iterative development approaches have proliferated in the field of environmental modelling. Similar to component-based approaches, parallels to software engineering practices may also be found. Iterative practices define phases in the development process – distinct steps through which model developers iteratively progress. Evidence of iterative software development practices can be found as far back as 1970 (in Royce, 1970) although the suggestion made therein was limited to two full iterations (“attempt to do the job twice”). Modern iterative development approaches, in both software and model development communities, suggest that each phase or step in the development process be revisited as often as necessary to achieve a desired outcome (Ambler, 2002; Jakeman et al., 2006). Iterative approaches suit scientific modelling as scientists often do not, and cannot, know in advance the implications and effects of intermediate findings produced during the course of the modelling process (Wilson et al., 2013).

Stakeholders are engaged in both software and model development cycles to gain insights and inform the development process. The exact nature of the model/software requirements becomes clearer through this process and may evolve from its initial specification. Modelers refer to this practice as participatory modelling and recognize the importance of the joint learning process such approaches enable (Arnold, 2016). Stakeholders, for example, may have differing and conflicting viewpoints which influences the scope of the model and how analysis of model results is conducted (Hamilton et al., 2015).

Stakeholder participation in the development process is now seen as a fundamental requirement to ensuring the software or model is fit-for-purpose (Ambler, 2002; Beck et al., 2001; Jakeman et al.,
In both software and model development fields, iterative participatory development can be viewed as an approach to managing the complexity of the final computational system in relation to its requirements (i.e. the objectives and purpose). Such approaches place greater emphasis on rapid prototyping and exploration than on the delivery of concrete milestones (Verweij et al., 2010).

2.3 Component-based Development

Component models are compartmentalized implementations of a model which are ‘wrapped’ with an interface for use with an MDF (Malard et al., 2017; Peckham et al., 2013). In this manner, the implementation details are abstracted away from other models and the MDF itself. This is described as ‘loose coupling’ in software engineering and refers to the fact that component models are not dependent on other component models or the MDF itself. Code changes to a loosely coupled model that do not affect the interface (in other words, the expected inputs and outputs) should not then require changes to other component models. This contrasts with tightly coupled components wherein changes to one may necessitate changes to another.

Hypothetically, a component model can be reused across MDFs so long as the interface specifications are compatible. Such ‘wrapped’ models can then be treated as independent, reusable building blocks from which a representation of an environmental system can be constructed to address a specific modelling need (Elag and Goodall, 2013). This compositional approach was adopted from the software engineering field where the idea first appeared in 1968, itself borrowed from approaches used in other engineering fields (Vale et al., 2016). Proponents of component-based integration cite many benefits to this approach including increased productivity, cost efficiency and software quality (Vale et al., 2016). Such benefits have been recognized in IEM and use of component-based approaches in IEM development has proliferated (see for example, Elag and Goodall 2013; de Kok et al. 2015; Kneis 2015; Malard et al. 2017).

Effective reuse of existing materials is key to reducing duplication of work across the modelling communities. Treating models as separate (and separable) components enables multiple teams to work concurrently on components for the same IEM without interfering in the development of other domain specific models. This separation of concerns is especially beneficial in participatory, iterative development approaches, as models are continually improved on, and adjusted in light of new information.

3 ENHANCING IEM DEVELOPMENT

As stated in the introduction, models (computational as opposed to conceptual) are software for a scientific purpose. Software development is a complex endeavor in and of itself. Further complexity is brought about by the methodologies introduced in the previous section which induce (ideally) frequent change to model components. Software engineers have developed approaches and methods to manage the development complexity, which may prove beneficial to the IEM development process. These involve fostering effective collaboration, testing, and the use of version control.

3.1 Reviewing Code

A common recurring theme in the literature is the importance of SD practices and effective collaboration between teams and stakeholders in the IEM development process. In reviewing the use of OpenMI, Knapen et al. (2013) suggests that “sufficient understanding of software development and its principles has to be available, or be developed before to make a serious effort in model integration”. Similarly, David et al. (2013) suggests that awareness of software ‘best practices’ should be fostered prior to the adoption of an MDF, including defining software coding standards and performing peer reviews of code. Development of an IEM is therefore contingent on the availability and application of technical SD expertise and effective collaboration between system modelers (as in de Kok et al. 2015).

Current approaches to IEM development rely on teams of researchers who are responsible for the development of a model in a specific aspect of the integrated system, acting as compartmentalized specialists. In this paradigm researchers not only focus on the science to the detriment of software
development principles (as in Ahalt et al., 2014; Sletholt et al., 2012) but their individual focus remains on their scientific specialty. This may explain, at least in part, issues and errors in IEM development that stem from a mismatch in assumptions and understanding during component model development (Belete et al., 2017).

These implementation issues cannot be resolved using an MDF or an interface specification based on a unified ontological framework. Individual modelers too focused on their own area of expertise may produce a component model that is difficult to integrate or unsuitable for use within an integrated context. Even if technical coupling of models is achieved, the resulting IEM may still produce unexpected results (Voinov and Cerco, 2010).

One suggestion to foster greater cross-domain collaboration is for IEM developers to conduct code reviews, a common practice in software development. By reviewing each other’s code, greater appreciation of the integrated system is gained, and faulty assumptions and errors in the code can be identified and corrected earlier in the process (Verweij et al., 2010). Critical analysis of the code is not limited to correctness of execution but also extends to its readability. Code could be judged on its clarity based on whether the reviewer with an understanding of the overall context, but with limited knowledge of the code purpose, can quickly determine the intent of the code.

Ultimately the code review process is intended to increase confidence in the model, improves code quality and readability, and transparency of the model between participants. These benefits serve to enhance understanding of how the interconnected system is represented by the IEM to both the developers and external reviewers. Readers are directed to Wilson et al. (2013) for further discussion and examples of code readability within the scientific programming context.

### 3.2 Code and Model Testing

At this point in the narrative, we have IEM developers who are engaged in iterative, participatory, and component-based development. Code reviews correct faulty assumptions that may have implications across domain boundaries and facilitates greater system understanding. Models are continually improved on, incorporating new knowledge and stakeholder feedback, resulting in the production of new versions of each component model. A change may induce unintentional side-effects which may propagate through the IEM and affect its behavior and results, even if individually they are technically sound and calibrated. How then do we verify that the code and the model are behaving as intended? If they are, did the changes measurably improve model performance according to criteria of interest (e.g. runtime, reduced complexity whilst maintaining precision and/or accuracy, and so on)?

Without incorporating testing practices in the IEM development process the answer to these may remain unknown. Major errors in research and retraction of published work has occurred due to a lack of testing and quality assurance practices in the sciences (Ahalt et al., 2014). Verifying that the model implementation behaves correctly in software is an ongoing issue in computational environmental science (Laniak et al., 2013). Because of our collective reliance on models, and the important role they play in policy development, testing practices must become further entrenched if IEM science is to progress further.

It is not suggested here that testing practices in the environmental sciences are non-existent, or its value unrecognized. MDFs may include some testing capabilities or otherwise have separate utilities available (e.g. David et al., 2013; White et al., 2014). Testing methodologies are, however, reportedly underutilized in IEM development processes. This may be due to time or cost constraints (Belete et al., 2017; Laniak et al., 2013; Voinov and Cerco, 2010), lack of training and experience or institutional reasons (Ahalt et al., 2014; Sletholt et al., 2012).

Uncertainty and sensitivity analysis (UA/SA) when applied as a method to analyze and verify model behavior have some parallels to SD testing practices. In a typical testing approach, the software (in its entirety, or a specific functionality) is run after a change and results compared against known expected values. Deviation from expected values then signifies a change has occurred (which may or may not be intended). Further description of SD testing practices, and their use in environmental modelling, may be found in Belete et al. (2017) and the references therein.
UA/SA methods could be applied in a similar manner to produce statistical measures which could be used to assess model performance and behavior (Saltelli and Annoni, 2010; Wirtz and Nowak, 2017). Comparing these across versions of component models and resulting IEMs may be a useful assessment approach that guides model development and manages complexity. Through this process a baseline comparison is developed and continually updated. IEM developers however, rarely conduct such analyses to an appropriate degree (Saltelli and Annoni, 2010; Wirtz and Nowak, 2017). This may be due to the inherent complexity of IEMs and the lack of resources available to conduct appropriate UA/SA investigations (Jakeman et al., 2006).

A potential issue here is the additional time necessary to develop tests and testing infrastructure, which places further pressure on IEM developers. Testing of an IEM, in its entirety, may be inappropriate and so the extent to which testing methodologies are applied is dependent on context. It is also important to emphasize that testing of code itself does not eliminate improper model behavior under all conditions. This is related to the understanding held by modelers that model behavior can only be verified to be consistent under a set of conditions. The validity of model results under all conditions can never be confirmed. Therefore, a model that clears all tests is only valid for the subset of possible conditions it is tested for and should not be assumed to be valid under all conditions. It is commonly accepted that no software can be determined to be completely free of defects. The goal, rather, is to determine a level of quality that is ‘good enough’ for its purpose.

Nevertheless, in large complex projects the use of tests allows IEM developers to focus on the development process while having some assurance that unintended changes were not introduced. These tests can be run in an automated manner, ideally at some consistent frequency whether that be after a change, daily, or weekly as the situation warrants. In doing so, IEM developers can be alerted to undesirable side-effects and their source, smoothing the model development process.

### 3.3 Versioning Code and Data

Throughout the model development process many versions of code can be produced for each model component, over each iteration, and may continue to change after its initial release. To manage these, software developers rely on version control systems. Such version control systems have been available for some time and evidence of their use in environmental model development can be found through the literature (e.g. Holzworth et al., 2014; Verweij et al., 2010). Widespread use, however, cannot be said to have been achieved as their use continues to be recommended (as in Belete et al., 2017; Hamilton et al., 2015; Knapen et al., 2013).

While there are many kinds of version control systems available, the general role it plays in the development process is to store all versions of code submitted to it. To give a simplified overview, developers work on a copy that is localised to their machine. Once an acceptable state is reached this copy is then submitted to a central database referred to as a “repository”. Further testing and code reviews may be conducted with the submitted code at this point. Moving back and forth seamlessly between different versions, or a previous version at a point in time, is also possible. Such flexibility enhances developer collaboration by allowing work to be conducted concurrently on the same codebase without fear of accidentally overwriting changes made by another. The same version control principles that apply to code can also be applied to data. Models through their evolution may expect data in different amounts, detail, formats, and so on. For this reason, data must be subject to the same versioning standards as code.

Through the disciplined use of version control, users can quickly identify the series of changes that led up to an issue detected by, for example, the testing process described above. Not only is this historic progression discoverable, but information on the user(s) who submitted the changes, when, and for what reason is also available. This information gives context on the intent of the code change which may be beneficial in resolving the cause of the identified issue. Use of a central repository simplifies the distribution process both between developers and, if publicly accessible, release of software. Such practices serve to document the code, its development, and the intended purpose, further increasing transparency.

### 3.4 Reusability and Reproducibility
The development of component models is ostensibly beneficial as these models can then be reused and reapplied across research contexts. Reusability does not, however, imply universal applicability. A component model may not be suitable for reuse, even if integration is technically feasible, due to choices made during its development that render it too context specific. In such cases it may be simpler to re-implement the model. This may be unachievable, however, due to issues with model documentation and accessibility of the underlying code (Parmar et al., 2016). Such issues may persist even if the original developer(s) are willing to assist (De Vos et al., 2011; Stodden et al., 2018).

Recall from the introduction that our chief concern here is to facilitate quicker progression through the environmental management cycles. Version control and accessible availability of model implementations is a key aspect towards increasing the speed and clarity of communication between researchers. From this perspective a lack of reusability may not be as significant an issue as it may first appear. Researcher’s primary aim is to conduct science, not to write software (Sletholt et al., 2012). Reusability is often not a primary goal, or even a desirable trait, as the necessary code to support the desired flexibility can further complicate the code and may be unrelated to the scientific concern at hand. In such situations, transparency of model intent may be of greater import than reusability.

Regardless of its purpose, a model should be transparent enough to facilitate re-implementation, at the very least to enable independent evaluation. This goes to the core of science; reproducibility (Hut et al., 2017; Hutton et al., 2016). The inability to reproduce model results has been identified as a problematic concern in recent IEM literature (De Vos et al., 2011) and indeed across the computational sciences in general (e.g. Hut et al., 2017; Hutton et al., 2016; Ince et al., 2012; McNutt, 2014; Saltelli and Funtowicz, 2017; Stodden et al., 2018).

Here, care must be taken not to conflate a model’s conceptual and mathematical specification with its implementation as software. A sufficiently described model may be re-implemented but does not guarantee that implementations will be equivalent. The original modelers may have made (undocumented) choices and assumptions which would result in differences between two implementations. This then may impart differences in model behaviour. For this reason, a bespoke non-reusable solution that is readily available (with the requisite data) is often more transparent than relying solely on published descriptions. Such single purpose, component models may be more appropriate to the modelling task than attempting to force the reuse of an existing component model. Thus, the additional level of transparency ultimately achieves the same goal as reusable component models. To clarify, it is not suggested here that reusable component models are unnecessary, rather that there may be circumstances which warrant the use of an alternative approach.

This view is perhaps optimistic as full reproducibility requires sufficient documentation of the code and data, not just a functioning model. Exploration of code documentation standards as used in software development circles is not in the purview of this paper, suffice to say that such standards should be adopted and applied in the scientific community (Ince et al., 2012; Knapen et al., 2013; Sletholt et al., 2012).

A lack of incentive for model developers to be transparent is cited as one possible reason for the lack of model reusability (De Vos et al., 2011). In recent times, publishers have begun to mandate that sufficiently detailed examples of a model accompany a submitted manuscript, along with the necessary data to run the model. While this is a positive step towards reproducibility, whether this is sufficient incentive remains to be seen.

4 CONCLUSION

Writing software is now a fundamental activity in the environmental sciences, and indeed in any computational science. Models are software for a scientific purpose, yet many beneficial software development practices appear not to have been generally adopted by IEM practitioners. The aim of this paper was to introduce some of these beneficial practices to model developers. These presented practices, from code reviews, testing, and managing code were chosen as they represent possible methods to overcome barriers that prevent quick(er) iteration of the management cycle. The fundamental aspect of these practices revolves around communication. Clearer code enables quicker peer review of code. Testing increases confidence in the code. Version control and repositories allow
for greater transparency of the development process and increases accessibility. These practices are compatible and complementary to existing IEM development approaches.

ACKNOWLEDGMENTS

The primary author is supported through an Australian Government Research Training Program (RTP) Scholarship. The authors would like to thank the reviewers for their valuable comments and suggestions to improve the quality of this paper.

REFERENCES


Hut, R.W., van de Giesen, N.C., Drost, N., 2017. Comment on “Most computational hydrology is not reproducible, so is it really science?” by Christopher Hutton et al.: Let hydrologists learn the latest computer science by working with Research Software Engineers (RSEs) and not reinvent the waterwheel our. Water Resour. Res.
Wilson, G., Aruliah, D., Titus Brown, C., Chue Hong, N.P., Davis, M., Guy, R.T., Haddock, S.H., Huff,