3-28-2011

Environmental Modeling Framework Invasiveness: Analysis and Implications

Wes J. Lloyd
University of Washington Tacoma, wlloyd@uw.edu

Olaf David

James C. Ascough II

Ken W. Rojas

Jack R. Carlson

See next page for additional authors

International Environmental Modelling and Software Society (iEMSs)
2010 International Congress on Environmental Modelling and Software
Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada
David A. Swayne, Wanhong Yang, A. A. Voinov, A. Rizzoli, T. Filatova (Eds.)
http://www.iemss.org/iemss2010/index.php?n=Main.Proceedings

Environmental Modeling Framework Invasiveness: Analysis and Implications

W. Lloyd [a], O. David [a], J.C. Ascough II [b], K.W. Rojas [c], J.R. Carlson [c], G.H. Leavesley [a], P. Krause [d], T.R. Green [b], L.R. Ahuja [b]

[a] Dept. of Civil and Environmental Engineering and Dept. of Computer Science, Colorado State University, Fort Collins, CO 80523 USA (e-mail address: odavid@colostate.edu)
[b] USDA-ARS, ASRU, 2150 Centre Ave., Bldg. D, Suite 200, Fort Collins, CO 80526 USA
[c] USDA-NRCS, 2150 Centre Ave., Bldg. A, Fort Collins, CO 80526 USA
[d] Department of Geography, Friedrich-Schiller-Universität Jena, Jena, Germany

Abstract: Environmental modeling frameworks support scientific model development by providing an Application Programming Interface (API) which model developers use to implement models. This paper presents results of an investigation on the framework invasiveness of environmental modeling frameworks. Invasiveness is defined as the quantity of dependencies between model code and the modeling framework. This research investigates relationships between invasiveness and the quality of modeling code. Additionally, we investigate the relationship between invasiveness and two common framework designs (lightweight vs. heavyweight). Five metrics to measure framework invasiveness were proposed and applied to measure invasiveness between model and framework code of several implementations of Thornthwaite and the Precipitation-Runoff Modeling System (PRMS), two hydrological models. Framework invasiveness measurements were compared with existing software metrics including size (lines of code), cyclomatic complexity, and object-oriented coupling with generally positive correlations being found. We found that models with lower framework invasiveness tended to be smaller, less complex, and have lower coupling. In addition, the lightweight framework implementations of the Thornthwaite and PRMS models were less invasive than the heavyweight framework model implementations. Our initial results suggest that framework invasiveness is undesirable for model code quality and that lightweight frameworks may help reduce invasiveness.

Keywords: Component-based modeling; Environmental modeling frameworks; Invasiveness; Frameworks; Software metrics.

1. INTRODUCTION

Environmental modeling frameworks support model development through provisioning of libraries of core modeling modules or components, component interaction/communication, time/spatial stepping/iteration, up/downscaling of spatial data, multi-threading/multiprocessor support, and cross language interoperability, as well as reusable tools for data analysis and visualization. Environmental modeling frameworks provide structure for models by supporting the disaggregation of modeling functions into components, classes, or modules. In this paper, we refer to functional units of model code as components. Components are able to be reused in other models coded to the same framework with little migration effort. One advantage of using an established environmental modeling framework is they often provide pre-existing libraries of components to help facilitate model development (Voinov et al., 2004; Argent et al., 2006). In this paper, we define the degree of dependency between an environmental modeling framework and model code as "framework invasiveness." This is the degree to

which model code is coupled to the underlying framework. Framework to application invasiveness occurs from the following:

- Use of a framework Application Programming Interface (API) consisting of data types and methods/functions which developers interface with to harness framework functionality;
- Use of framework specific data structures (e.g., classes, types, constants);
- Implementation of framework interfaces and extension of framework classes;
- Boilerplate code ("non-science" code required for model to run under the framework); and
- Framework requirements including language, platform, and libraries.

Framework to application invasiveness is a type of code coupling; object-oriented coupling (i.e., coupling between classes in an object-oriented program) has been shown to correlate inversely with software quality (Briand et al., 2000; Basil et al., 1996). One goal of this research is to explore relationships between environmental model code quality and the degree of invasiveness between model code and environmental modeling frameworks. There are many dimensions to model code quality, often referred to as quality attributes. Quality attributes that may be impacted by framework invasiveness include understandability, maintainability, and portability/reusability.

Modeling frameworks can be classified as either heavyweight or lightweight (Richardson, 2006). Framework type characteristics are described in Table 1. A primary difference is how frameworks present functionality to the developer. Heavyweight frameworks provide developers with a large application programming interface (API) and developers typically spend considerable time becoming familiar with it before writing model code. Lightweight frameworks provide functionality to developers using techniques aimed at reducing the API's overall size. Programming annotations capture metadata and are used to identify points in the model code where framework functionality should be integrated. Framework integration can also be accomplished using external XML files. Wherever possible, "convention over configuration" is favored in that system defaults are assumed and developers only specify unconventional details in model code. Non-default behavior may include unique component data input/output requirements, pre-conditions, post-conditions, etc. in model code. Framework specific data types which take the place of system data types are avoided in lightweight framework designs. A second goal of this research is to explore the relationship between the framework type (i.e., heavyweight vs. lightweight) and the degree of framework to application invasiveness.

Table 1. Heavyweight versus lightweight framework design classification.

Heavyweight Frameworks	Lightweight Frameworks

• Components under the framework:
  ▪ bound statically at compile time
  ▪ tightly coupled to the framework by extension of framework classes, implementation of framework interfaces, use of framework data types, and use of framework functions
• Provides specialized versions of native language data types
• Have a "large" programming interface (API)
• Use may depend on many libraries
	• Components under the framework:
	  ▪ bound dynamically at run time by use of language annotations/dependency injection (inversion of control software design pattern)
	  ▪ loosely coupled and framework independent
	• Convention over configuration: developers only specify unconventional details in code as defaults are assumed
	• Uses native language data types
	• Have a "small" programming interface (API)

The broad objectives of this research are to investigate the implications of framework invasiveness on model code quality and to investigate the framework invasiveness characteristics of both lightweight and heavyweight frameworks. Specifically, we seek to answer the following research questions: 1) what is the impact of framework to model code invasiveness on model code quality, and 2) do the design characteristics of lightweight frameworks enable model development resulting in lower framework to model code invasiveness? Previously, environmental modelers have developed models with only a vague understanding of how the design of environmental modeling frameworks impact modeling efforts. A better understanding of the phenomenon of framework to application

invasiveness can help modelers in choosing and designing modeling frameworks to improve the quality of scientific models throughout their entire software life-cycles.

To investigate the above questions, we performed a case study using two environmental models, a monthly water balance model (Thornthwaite) and a complex watershed-scale model (the Precipitation Runoff Modeling System, PRMS). A set of software metrics was devised and applied to quantify the invasiveness between the framework and model code. Several traditional software quality metrics were used to assess the quality of the environmental model implementations in terms of size, complexity, and object-oriented coupling. An analysis of results was performed to identify relationships between model code quality and invasiveness, and also between framework type (e.g., heavyweight/lightweight) versus invasiveness.

2. ENVIRONMENTAL MODELING FRAMEWORKS

For this framework invasiveness study, the ESMF 3.1.1, CCA 0.6.6, OpenMI 1.4, and OMS 2.2 and 3.0 environmental modeling frameworks were used to implement the Thornthwaite (Thornthwaite, 1948) and PRMS (Leavesley et al., 2006) environmental models. Additionally, three non-framework based implementations of Thornthwaite were implemented in Java, C++, and FORTRAN to assist in developing framework-based versions. ESMF is an open source framework developed by the National Center for Atmospheric Research (NCAR) for building climate, numerical weather prediction, data assimilation, and other Earth science software applications (Collins et al., 2005). CCA was developed by the members of the Common Component Architecture Form, and is a component architecture for high performance computing (Armstrong et al., 1999). OpenMI is sponsored by the European Commission LIFE Environment program and is a software component interface definition for developing models in the water domain (Blind and Gregersen, 2005). OpenMI Thornthwaite model implementation in this study was performed using a Java-based implementation of OpenMI, although a .NET/C# version exists and is generally considered more popular. The Object Modeling System (OMS) versions 2.2 and 3.0 are developed by the USDA – Agricultural Research Service (ARS) in cooperation with Colorado State University. OMS facilitates component-oriented simulation model development in Java, C/C++ and FORTRAN (David et al., 2002), and version 2.2 provides an integrated development environment (IDE) with numerous tools supporting data retrieval, GIS, graphical visualization, statistical analysis and model calibration (Ahuja et al., 2005). The ESMF 3.1.1, CCA 0.6.6, OpenMI 1.4, and OMS 2.2 frameworks can be considered as heavyweight frameworks where modeling code is coupled to the framework through dependencies on a framework's API, i.e., components must use specific data types and functions to interface with the framework. OMS 3.0 has been developed with a “non-invasive” lightweight framework design for model development. That is, modeling components have been decoupled from the framework API wherever possible so that they exist as plain classes implementing only model specific logic. In addition, boilerplate code has been re-factored out using language annotations.

3. ENVIRONMENTAL MODELS

For this study, we investigated several implementations of Thornthwaite and the Precipitation-Runoff Modeling System (PRMS) hydrological models. Thornthwaite is a monthly water balance model which simulates water allocation among components of a hydrological system (Thornthwaite, 1948). The model was selected since it has a typical structure for a hydrological simulation model and its size and complexity were manageable for this study. All model implementations were coded to produce identical numeric output, programming language specific formatting functions were not used, and only framework support for component aggregation and component interaction/communication were utilized. The average code size of the framework based Thornthwaite model implementations was 754 lines of code (LOC).

The Precipitation-Runoff Modeling System (PRMS) is a deterministic, distributed-parameter model developed to evaluate the impact of various combinations of precipitation, climate, and land use on stream flow, sediment yields, and general basin hydrology (Leavesley et al., 2006). PRMS was implemented in Java using the OMS 2.2

and 3.0 frameworks as time and resources were lacking to implement the model under additional frameworks. The PRMS implementations utilized framework support for component aggregation, interaction and communication as well as model time stepping. The average implementation size of the PRMS models studied was 13,580 LOC.

4. FRAMEWORK INVASIVENESS MEASURES

Research in object-oriented software evaluation has produced numerous metrics which help to measure attributes such as the coupling, cohesion, and inheritance among classes in an object-oriented program (Chidamber and Kemerer, 1994). However, the existing metrics were not designed to specifically quantify the dependencies between framework and modeling. The measures in the following sections were applied to quantify invasiveness between environmental modeling frameworks and model code.

4.1. Framework Data Types (FDT) and Framework Functions (FF)

We quantify usage of two primary framework constructs in a model: framework data types and framework functions. We count the total number of framework data types (classes, data structures, types, etc.) used (FDT-used), and the total number of uses of these framework data types in modeling code (FDT-uses). The total number of framework functions (functions, methods, subroutines, etc.) used (FF-used), and the total number of uses of these framework functions (calls) appearing in the modeling code (FF-uses) are counted. Three variations of the framework metrics were calculated: a raw count, a count of framework construct usage weighted per 1000 lines of code (KLOC) (e.g. FDT/FF-used/-uses per KLOC), and the percentage of usage relative to all framework constructs used/uses in the application code (e.g. % FDT/FF-used/-uses).

4.2. Framework Dependent Lines of Code (FDLOC)

To measure the invasiveness between model code and framework code, we counted the total number of lines of code which depend on the framework. A framework dependent line of code is defined as a line of code which depends on the framework such that if the framework libraries were removed the line would not compile. This implies that a framework dependent line of code contains at least one framework reference. In this study, we calculated two variations of FDLOC: raw count and a percentage relative to the total lines of model code (% FDLOC).

4.3 Software Quality Measures

As a surrogate for measuring model quality, we used three measures in this study: 1) size, measured by counting lines of code (LOC); 2) complexity, measured by determining cyclomatic complexity; and 3) coupling, measured using efferent coupling (fan-out), and afferent coupling (fan-in). Cyclomatic complexity (CC) counts the number of linearly independent paths through a program's source code. This is a surrogate for measuring code complexity and has been a widely used in computer science. To measure coupling, we used both efferent and afferent coupling measures because they can be collected on programs in both procedural and object-oriented languages. Efferent coupling is the number of classes which make reference to a class. This can be thought of as the number of uses "outside" of the class. Afferent coupling is a dependency measure which counts the number of classes referenced by a class. This can be thought of as the classes used "inside" the class. Size, complexity and coupling measures generally inversely correlate with code quality (Basil et al., 1996; Briand et al., 2000; and Briand et al., 1999).

5. RESULTS

Static analysis tools were used to support analysis of the model implementations. SLOCCOUNT (SLOCcount, 2009) was used to count lines of code. Understand 2.0 Analyst (Understand, 2009) was used to collect the LOC, cyclomatic complexity, coupling between objects (CBO), and fan-in/fan-out coupling software metrics. Function and data type usage reports produced by Understand 2.0 were parsed using a custom program to generate data for the FDT and FF usage measurements. FDLOC were determined manually by counting lines of code.

5.1 Thornthwaite Model

The Thornthwaite model implementations were coded to provide identical output given the same inputs to allow us to attribute differences observed between the implementations to the invasiveness incurred from the different frameworks. Size and complexity measurements of the Thornthwaite model framework implementations are shown in Table 2.

Table 2. Thornthwaite model size and complexity metrics.

Language/ Framework	Total LOC	Average CC/method	Total CC
FORTRAN only	244	3.33	40
OMS 3.0 Java	295	2.38	31
Java only	319	2.85	37
C++ only	405	2.41	41
OMS 2.2 Java	450	1.18	103
ESMF 3.1.1 C	583	1.97	65
ESMF 3.1.1 FORTRAN	683	1.44	56
OpenMI 1.4 Java	880	1.61	116
CCA 0.6.6 Java	1635	2.25	276

The OMS 3.0 framework was the only framework which enabled a smaller model (in LOC) than the implementation in the equivalent native language, i.e., the OMS 3.0 Thornthwaite implementation was 295 LOC compared to 319 for Java-only. Ideally, a framework-based model implementation should have a smaller code size than a plain-language implementation with the reduced model code size reflecting code reuse where some aspects of the model implementation are provided by framework code.

Coupling measures for the Thornthwaite model framework implementations are shown in Table 3. For the Thornthwaite model framework implementations, measurements for size, complexity and coupling were positively correlated. Total LOC and cyclomatic complexity had a correlation coefficient of $r = 0.94$ ($df = 4$, $p < 0.01$), total LOC and total fan-in had a correlation coefficient of $r = 0.92$ ($df = 3$, $p < 0.05$), and total cyclomatic complexity with total fan-in had a correlation coefficient of $r = 0.95$ ($df = 3$, $p < 0.02$).

Table 3. Thornthwaite model coupling measures.

Language/Framework	Total Fan-In (Afferent)	Total Fan-Out (Efferent)
OMS 3.0 Java	116	70
OMS 2.2 Java	116	70
ESMF 3.1.1 C	100	155
ESMF 3.1.1 FORTRAN	N/A	N/A
OpenMI 1.4 Java	126	177
CCA 0.6.6 Java	195	215

Detailed invasiveness measurements for the Thornthwaite model framework implementations are shown in Table 4. For the framework invasiveness measures, the OMS 3.0 Thornthwaite model framework implementation appeared to be the least invasive, i.e., this implementation had far fewer framework dependencies than others. The Thornthwaite scientific code was essentially the same for all of the environmental modeling framework implementations, with the observed differences resulting from various framework-specific requirements to implement the model. The large variations in the metrics suggest that variations in framework design likely impact the modeling code. Table 4 also shows framework invasiveness metrics scaled to a percentage. The percentage scaling shows how much of the overall percentage of an attribute is framework dependent. Overall, a model implementation with low framework invasiveness should have a low percentage of data type, functions, and LOC dependence on the underlying framework.

The final invasiveness measurement scaling shown in Table 5 is a scaling of attribute occurrences per 1000 lines of code (KLOC), i.e., this scaling represents the expected number of occurrences if there were 1000 lines of code. Since the model implementations

varied in size, this scaling provides a method for a side-by-side comparison. FDLOC, FDT-used, FF-used correlated with model size (df = 4, p<.05); however, none of the percentage or scaling invasiveness measures correlated with size. For complexity, three invasiveness measures (FDLOC, FDT-used, and FF-used) were shown to correlate with total cyclomatic complexity (df = 4, p<.05). A correlation existed between FF-used/KLOC and average method cyclomatic complexity; however, correlation coefficients for other measures with average CC/method seem almost random so it is possible the FF-used/KLOC relation is spurious. Correlation coefficients between invasiveness and total complexity were generally positive though they varied in magnitude. Total fan-in (afferent) and fan-out (efferent) coupling correlated significantly with FDLOC, FDT-used, and also %FF-used (fan-in only) (df=3, p < 0.05).

Table 4. Framework invasiveness detailed measurements**.**

Implementation	FDLOC	FDT-used	FDT-uses	FF-used	FF-uses
OMS 3.0 Java	44	1	1	8	21
OMS 2.2 Java	147	5	72	7	33
ESMF 3.1.1 C	178	10	122	13	77
ESMF 3.1.1 FORTRAN	280	3	109	11	148
OpenMI 1.4 Java	338	8	73	20	280
CCA 0.6.6 Java	533	15	135	48	215
Implementation	FDLOC (%)	FDT-used (%)	FDT-uses (%)	FF-used (%)	FF-uses (%)
OMS 3.0 Java	14.84	4.67	1.35	26.67	40.38
OMS 2.2 Java	32.67	41.67	64.29	50.00	73.33
ESMF 3.1.1 C	30.85	30.30	49.59	46.43	76.24
ESMF 3.1.1 FORTRAN	41.42	27.27	51.90	78.57	96.10
OpenMI 1.4 Java	38.41	23.53	32.30	37.74	79.10
CCA 0.6.6 Java	32.60	46.88	49.82	70.59	69.58
Implementation	FDLOC/KLOC	FDT-used/KLOC	FDT-uses/KLOC	FF-used/KLOC	FF-uses/KLOC
OMS 3.0 Java	148	3.39	3.39	27.12	71.19
OMS 2.2 Java	327	11.11	160.00	15.56	73.33
ESMF 3.1.1 C	309	17.15	209.26	22.30	132.08
ESMF 3.1.1 FORTRAN	414	4.39	159.59	16.11	216.69
OpenMI 1.4 Java	384	9.09	82.95	22.73	318.18
CCA 0.6.6 Java	326	9.17	82.57	29.36	131.50

5.2 PRMS Model

The invasiveness metrics were applied to evaluate the PRMS model implementations under the OMS 2.2 and 3.0 frameworks. Size and complexity metrics are shown in Table 5. PRMS model code size was reduced 40% in the OMS 3.0 framework implementation. Much of the size reduction can be attributed to the elimination of component getter and setter methods. Getter and setter methods are accessor methods which intercept read/write access to data variables in an object-oriented program. These constructs are encouraged to provide data encapsulation to prevent unintentional changes to variables. The average complexity per method increased significantly from OMS 2.2 to OMS 3.0. This is because the total number of methods dropped significantly through elimination of the getter and setter methods. A reduction in model complexity is reflected in the more than three-fold reduction in total cyclomatic complexity observed in the OMS 3.0 PRMS model implementation versus OMS 2.2 (Table 5).

Table 5. PRMS model size and complexity metrics.

Framework Implementation	Total LOC	Average CC/method	Total CC
OMS 3.0 Java	10163	9.75	702
OMS 2.2 Java	16997	1.37	2575

Table 6. PRMS model coupling measures.

Framework Implementation	Total Fan-In (Afferent)	Total Fan-Out (Efferent)	Avg. Number Methods/Class
OMS 3.0 Java	1232	755	3.6
OMS 2.2 Java	3517	1428	85.41

Coupling measures for the PRMS model implementations under the OMS 2.2 and 3.0 frameworks are shown in Table 6. Reductions in both total fan-out (efferent) and total fan-in (afferent) coupling are observed in the OMS 3.0 PRMS implementation. Coupling was likely reduced in relation to the reduction in code size attributed by removing getter and setter methods. The average number of methods per component dropped from 85 to 3.6 in OMS 3.0. Framework invasiveness measures for the PRMS model implementations are shown in Table 7. A significant reduction is seen in the use of framework data types and functions in the OMS 3.0 lightweight framework implementation.

Table 7. PRMS model invasiveness measures.

Implementation	FDT-used	FDT-uses	FF-used	FF-uses
OMS 3.0 Java	1	3	5	7
OMS 2.2 Java	16	1788	15	2854
Implementation	FDT-used (%)	FDT-uses (%)	FF-used (%)	FF-uses (%)
OMS 3.0 Java	5	0.19	5.5	2
OMS 2.2 Java	50	65.9	19.2	91.8
Implementation	FDT-used/KLOC	FDT-uses/KLOC	FF-used/KLOC	FF-uses/KLOC
OMS 3.0 Java	0.09	0.29	0.49	0.69
OMS 2.2 Java	0.94	105.2	0.88	167.9

6. DISCUSSION

To investigate relationships between model code quality and invasiveness, we used the approach recommended by Briand et al. (1999, 2000) and Basil et al. (1996) to use Chidamber and Kemerer's (1994) object-oriented software metrics as indirect measures of software quality. Using fan-in/fan-out coupling as an inverse surrogate of software quality, we found that framework implementations for both models having the lowest invasiveness measures for FDT-uses and FF-uses also had the lowest values for fan-in/fan-out coupling ($p = 0.002, 0.011$; $df = 5$). Framework implementations for both models with higher fan-in/fan-out coupling used more framework functions and data types. This relationship suggests that more invasive model implementations may exhibit lower code quality. We also found that model framework implementations with low invasiveness measures for FDT-uses and FF-uses also had the smallest code sizes (LOC) ($p = 0.024, 0.024$, $df = 5$) and total cyclomatic complexity (total CC) ($p = 0.0007, 0.0007$, $df = 5$). Models with larger LOC and total CC also used more framework functions and data types. The results indicate that framework-based model implementations which used the most framework functions and data types had larger size, complexity and more coupling. These relationships suggest a negative relationship between framework invasiveness and software quality.

The Thornthwaite and PRMS model implementations under the lightweight OMS 3.0 framework had lower framework to model invasiveness (Tables 4 and 7). Additionally, the Thornthwaite and PRMS model implementations under the OMS 3.0 framework had lower overall code size (LOC), cyclomatic complexity, and afferent (fan-in) and efferent (fan-out) coupling (Tables 3 and 6). It appears that a lightweight framework based modeling approach produces both smaller and simpler model implementations.

7. SUMMARY AND CONCLUSIONS

This paper presents a unique comparison of environmental modeling framework invasiveness using the Thornthwaite and PRMS hydrologic models. Our results showed that less invasive model implementations tended to have higher code quality as observed in terms of code size, complexity and coupling. Models implemented using the OMS 3.0 had the lowest invasiveness scores and the smallest size, complexity, and coupling. For the Thornthwaite model, the OMS 3.0 implementation was on average 40% as large as the heavyweight framework implementations and about 30% as complex. For the PRMS model, the OMS 3.0 implementation was 40% smaller and about 30% as complex as the heavyweight OMS 2.2 framework implementation. Overall, the OMS 3.0 framework produced less invasive model implementations when compared to the heavyweight

framework implementations using OMS 2.2, ESMF 3.1.1, OpenMI 1.4, and CCA 0.6.6. In conclusion, the lightweight framework approach to environmental modeling appears to produce smaller, less complex models with less coupling and framework-to-model invasiveness. Based on this result, a lightweight framework approach to environmental modeling appears to help modelers develop higher quality and more concise model implementations. For environmental modeling, this lightweight framework design approach deems further attention.

ACKNOWLEDGMENTS

We would like to acknowledge Cecelia DeLuca from UCAR, a member of the ESMF support team, who provided a code review of the ESMF FORTRAN Thornthwaite model implementation and also Dr. Andrea Antonello who provided a code review of the OpenMI 1.4 Java Thornthwaite implementation.

REFERENCES

Ahuja, L.R., Ascough II, J.C., and David, O., Developing natural resource modeling using the object modeling system: feasibility and challenges. *Advances in Geosciences*, 4, 29-36, 2005.

Argent, R.M., Voinov, A., Maxwell, T., Cuddy, S.M., Rahman, J.M., Seaton, S., Vertessy, R.A., and Braddock, R.D., Comparing modelling frameworks – A workshop approach. *Journal of Environmental Modelling & Software*, 21 (7), 895-910, 2006.

Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., and Smolinski, B., Toward a common component architecture for high-performance scientific computing. Proceedings of the 8th Intl. Symposium on High Performance Distributed Computing, pp. 115-124, 1999.

Basil, V.R., Briand, L.C., and Melo, W.L., A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22 (10), 751-761, 1996.

Blind, M., and Gregersen, J.B., Towards an Open Modeling Interface (OpenMI) the HamonET project. *Advances in Geosciences*, 4, 69-74, 2005. .

Briand, L. C., Wust, J., Daly, J., and Porter, D.V., Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of Systems & Software*, 15 (3), 245-273, 2000.

Briand, L. C., Wiist, J., Ikonomovski, H. L., Investigating quality factors in object-oriented designs: an industrial case study. Proceedings of the 21st International Conference on Software Engineering (ICSE '99), pp. 345-354, 1999.

Chidamber, S.R., and Kemerer, C.F., A metrics suite for object oriented design. *Transaction on Software Engineering*, 20 (6), 476-493, 1994.

Collins, N., G. Theurich, C. DeLuca, M. Suarez, A. Trayanov, V. Balaji, P. Li, W. Yang, C. Hill, and A. da Silva, Design and Implementation of Components in the Earth System Modeling Framework (ESMF). International Journal of High Performance Computing Applications, Fall/Winter, 2005.

David, O., Markstrom, S.L., Rojas, K.W., Ahuja, L.R., and Schneider, W., The object modeling system. In: Ahuja L.R., Ma. L., Howell T.A. (Eds.). Agricultural system models in field research and technology transfer. Lewis Publishers, Boca Raton, FL, USA, pp. 317-344, 2002.

Leavesley, G.H., Markstrom, S.L., and Viger, R.J., USGS Modular Modeling System (MMS) - Precipitation-Runoff Modeling System (PRMS). In: Singh, V.P. and Frevert, D.K. (Eds.), Watershed Models. CRC Press, Boca Raton, FL, pp. 159-177, 2006.

Richardson, C., Untangling Enterprise Java. *ACM Queue* 5 (4), 33-44, 2006.

SLOCCount, David A. Wheeler. Available at http://www.dwheeler.com/sloccount/ (accessed March 2010).

Thornthwaite, C.W., An approach toward a rational classification of climate. *Geographical Review*, 38 (1), 55-94, 1948.

Understand – Source Code Analysis and Metrics, Scientific Toolworks, Inc. Available at http://www.scitools.com/ (accessed March 2010).

Voinov, A., Fitz, C., Boumans, R., Costanza, R., Modular ecosystem modeling. Journal of Environmental Modelling & Software, 19 (3), 285-304, 2004.