18th World IMACS / MODSIM Congress, Cairns, Australia 13-17 July 2009
http://mssanz.org.au/modsim09

An exploratory investigation on the invasiveness of environmental modeling frameworks

Wes Lloyd [1], Olaf David [1], James C. Ascough II [2], Ken W. Rojas [3], Jack R. Carlson [3], George H. Leavesley [1], Peter Krause [4], Timothy R. Green [2], and Lajpat R. Ahuja [2]

[1] *Dept. of Civil and Environmental Engineering and Dept. of Computer Science, Colorado State University, Fort Collins, CO 80523 USA*

[2] *USDA-ARS, 2150 Centre Ave., Bldg. D, Suite 200, Fort Collins, CO 80526 USA*

[3] *USDA-NRCS, 2150 Centre Ave., Bldg. A, Fort Collins, CO 80526 USA*

[4] *Department of Geography, Friedrich-Schiller-Universität Jena, Jena, Germany*

Email: wlloyd@acm.org

Abstract: Environmental modeling frameworks provide an array of useful features that model developers can harness when implementing models. Each framework differs in how it provides features to a model developer via its Application Programming Interface (API). Environmental modelers harness framework features by calling and interfacing with the framework API. As modelers write model code, they make framework-specific function calls and use framework specific data types for achieving the functionality of the model. As a result of this development approach, model code becomes coupled with and dependent on a specific modeling framework. Coupling to a specific framework makes migration to other frameworks and reuse of the code outside the original framework more difficult. This complicates collaboration between model developers wishing to share model code that may have been developed in a variety of languages and frameworks.

This paper provides initial results of an exploratory investigation on the invasiveness of environmental modeling frameworks. Invasiveness is defined as the coupling between application (i.e., model) and framework code used to implement the model. By comparing the implementation of an environmental model across several modeling frameworks, we aim to better understand the consequences of framework design. How frameworks present functionality to modelers through APIs can lead to consequences with respect to model development, model maintenance, reuse of model code, and ultimately collaboration among model developers. By measuring framework invasiveness, we hope to provide environmental modeling framework developers and environmental modelers with valuable information to assist in future development efforts. Eight implementations (six framework-based) of Thornthwaite, a simple water balance model, were made in a variety of environmental modeling frameworks and languages. A set of software metrics were proposed and applied to measure invasiveness between model implementation code and framework code. The metrics produced a rank ordering of invasiveness for the framework-based implementations of Thornthwaite. We compared model invasiveness results with several popular software metrics including size in lines of code (LOC), cyclomatic complexity, and object oriented coupling. To investigate software quality implications of framework invasiveness we checked for relationships between the Chidamber and Kemerer (1994) object oriented software metrics and our framework invasiveness measures.

For the six framework-based implementations of Thornthwaite we found a five-fold variation in code size (LOC). We observed up to a seven-fold variation in total cyclomatic complexity, and a two to three-fold variation in object oriented coupling. For the model implementations we found that total size, total complexity, and total coupling all had a significant positive correlation. The raw count version of our invasiveness measures correlated with application size (LOC), total cyclomatic complexity, total efferent coupling (fan out) and total afferent coupling (fan in). Large size, complexity, and high levels of coupling between units (classes, modules) in a software system are often cited in software engineering as causes of high maintenance costs due to poor understandability and flexibility of the code. This study provides initial results but further investigation is desired to evaluate the utility of our invasiveness measurement approach as well as the software quality implications of framework invasiveness.

Keywords: *Environmental modeling frameworks, Invasiveness, Frameworks, Software metrics*

1. INTRODUCTION

A software framework is a set of cooperating classes that makes up a reusable design for a specific application domain. Frameworks define the architecture of applications by proving structure for partitioning functionality into classes and objects. Frameworks specify how classes and objects collaborate, and also manage the thread of control. Frameworks emphasize design reuse which leads to an inversion of control between the application and the software on which it is based (Gamma, 1995). Frameworks exist to support both business needs (domain specific) and computational functionality. Frameworks which provide computational function support non-business related functions such as database access, graphical interface development, transaction management, etc. Frameworks which service a problem domain provide business functionality for specific problem domains such as environmental modeling, industrial control systems, networking and telecommunications, inventory tracking, etc.

Environmental modeling frameworks provide support for developing and deploying environmental simulation models. Some of the functions provided include support for aggregation of models into components, component interaction/communication, time/spatial stepping/iteration, re-gridding of arrays or up/downscaling of spatial data, multithreading/multiprocessor support, and cross language interoperability. These frameworks frequently provide a mechanism for the disaggregation of model functionality into individual units, commonly referred to as components, classes, or modules. In this paper, we will refer to functional units as components. Components, once implemented in a particular framework, are able to be reused in other applications coded to the framework with no migration effort. One advantage of choosing a popular framework is that there is often a plethora of pre-existing components which can help facilitate application development.

A developer writes business code using a domain specific framework. When coding the application the developer couples business code to the framework through the use of the framework application programming interface (API) and framework specific data types. By using framework interfaces and data types the developer harnesses functionality provided by the framework to realize the application implementation. We shall call the dependency between the framework and business code "framework invasiveness." It is the degree to which the business code is coupled to the underlying framework. Framework to application invasiveness occurs from the following:

- Use of framework API methods/functions;
- Use of framework specific data structures (classes) and constants;
- Implementation of framework interfaces;
- Extension of framework classes;
- Boilerplate code;
- Framework requirements: language, platform, libraries required by the framework; and
- Organizational investment: training, financial, development.

Framework to application invasiveness is a type of code coupling, which is the degree to which program modules depend on each other. Object-oriented coupling (i.e., coupling between classes in an object-oriented program) has been shown to correlate inversely with software quality (Briand, 2000; Basil, 1996). A consensus among software developers is that a high degree of coupling to any framework or library may result in lower software quality. By measuring framework invasiveness we hope to better understand its implications with respect to software quality. Software quality attributes that framework invasiveness likely impacts include:

- Understandability - the ability for developers new to the code to understand the implementation;
- Maintainability - the ease of maintaining the code for bug fixes, feature enhancements, and upgrading to new framework versions (Dig, 2006); and
- Portability/Reusability - the ease of porting application code for use outside the framework or in other frameworks.

Frameworks can be classified as either heavyweight or lightweight (Richardson, 2006). Characteristics of framework design are described in Table 1. Framework invasiveness may also provide a means to evaluate framework design tradeoffs. We hypothesize that invasiveness of heavyweight frameworks will be higher than that of lightweight frameworks. In addition to understanding quality implications, framework invasiveness may be useful to quantify the development burden incurred by a programmer using a particular framework.

Table 1. Framework design classification

Heavyweight Frameworks	Lightweight Frameworks
- Framework overloads/wraps native language data types; - Framework has "large" programming interface (API); - Components under the framework are tightly-coupled to the framework; - Components bound to the framework statically at compile time; and - Framework use depends on many libraries.	- Framework uses native language data types; - Framework has a small programming interface (API); - Components under the framework are loosely-coupled and largely independent of the framework; and - Components bound to the framework dynamically using language annotations/dependency injection techniques (inversion of control - software design pattern).

2. INVASIVENESS MEASURES

Object-oriented coupling measures quantify coupling between classes in an object-oriented program (Chidamber and Kemerer, 1994). They include measures such as coupling between object classes (CBO), efferent coupling (fan-in), afferent coupling (fan-out), response for a class (RFC), and message passing coupling (MPC). But are these coupling measures useful to quantify the invasiveness of using a particular framework for application implementation? Static analysis tools which collect these metrics generally only measure the coupling between system classes. They do not measure coupling to compiled library classes where the source code is not present. New metrics are required to quantify the dependencies between the application code and a framework. We propose the following set of measures which can be used to quantify the dependencies between a framework and the application code.

2.1. Framework Data Types (FDT)

To measure the invasiveness between an application and a framework, we counted the total number of framework data types used (FDT-used), and the total number of framework data type uses in the application implementation (FDT-uses). FDT-used counts the total number of unique framework types (classes, data structures, types, etc.) which are used in the application. FDT-uses counts the total number of uses of these data types in the application. We calculated three variations of the FDT metric: raw counts, the number of occurrences per 1000 lines of code (KLOC) (FDT-used/-uses per KLOC), and the percentage of occurrences relative to all data types used/uses in the application code (% FDT-used/-uses). A raw count can be used to compare the number of framework types used/uses among different framework-based application implementations. By weighting FDT-used/-uses per KLOC, we can compare the density of framework data type usage between applications using a common code size. The percentage of FDT-used/-uses versus all data types used/uses in a program can help us to understand how much of the data footprint of the application depends on the framework. When counting total data types -used/-uses we do not include application specific types which may be declared to implement the application.

2.2. Framework Functions (FF)

To measure the invasiveness between an application and a framework we counted the total number of framework functions used (FF-used), and the total number of framework function uses (calls) appearing in the application implementation (FF-uses). FF-used counts the total number of unique framework functions (functions, methods, subroutines, etc.) which are called in the application. FF-uses counts the total number of calls to framework functions in the application. We calculated three variations of FFs: raw counts, the number of occurrences per 1000 lines of code (KLOC) (FF-used/-uses per KLOC), and the percentage of occurrences relative to all functions used/uses in the application code (% FF-used/-uses). The raw count can be used to compare the number of framework functions used/uses among different framework-based application implementations. By weighting FF-used/-uses per KLOC, the density of framework function usage between applications can be compared using a common code size. The percentage of FF-used/-uses versus all functions used/uses in a program can help us understand the ratio of function usage that depends on the framework. When counting total functions -used/-uses we do not include functions defined by the application to implement the application itself.

2.3. Framework Dependent Lines of Code (FDLOC)

To measure the invasiveness between an application and a framework we counted the total number of lines of code which depend on the framework. A framework dependent line of code is defined as any line of code which depends on the framework such that if the framework were removed the line of code would not

compile. Ideally, we would like to measure framework boilerplate code. Boilerplate code is defined as sections of code that have to be included in many places with little or no alteration. Framework boilerplate code is required by the framework for application implementation. However, it is very difficult to define precisely what a boilerplate line of code is for counting purposes because the classification requires a functional interpretation by the person or tool performing the count. Framework dependent lines of code can easily be identified because of the strict requirement that they contain a reference to the framework. In this initial study we calculated two variations of FDLOC: raw count and a percentage relative to the total lines of application code (% FDLOC). The raw count can be used to compare number of lines of framework dependent code between application implementations. The percentage of FDLOC versus all LOC in a program can be used to help understand what percentage of the code depends on the framework.

2.4. Framework Invasiveness Measurement

To compare the framework invasiveness of our model implementations, the three framework invasiveness dimensions above were aggregated into a single measurement. In order to combine the three dimensions captured by the measures equally FDT and FF measures were weighted at 50% each and FDLOC at 100%. To scale measurements for comparison, for each measure the average (avg) and standard deviation (stddev) among the set of n framework-based model implementations was calculated. Invasiveness metric values (FDT-used, FDT-uses, FF-used, FF-uses, FDLOC) were expressed as the number of standard deviations away from the average using the INV_Measure formula, and then aggregated together in the INV formula:

$$\text{INV\_Measure} = \frac{(\text{INV\_Measure}_n - \text{INV\_Measure}_{avg})}{\text{INV\_Measure}_{stddev}}$$

$$\text{INV} = \frac{\{(\text{FDT-used} + \text{FDT-uses}) * 0.5\} + \{(\text{FF-used} + \text{FF-uses}) * 0.5\} + \text{FDLOC}}{3}$$

Three invasiveness scores were calculated: raw measure values (INV raw), percentage of framework to non-framework usage (INV %), and density of framework usage weighted per 1000 lines of code (KLOC) (INV density/KLOC).

3. MODEL IMPLEMENTATION

In this study we implemented Thornthwaite, a simple monthly water balance model which simulates water allocation among components of a hydrological system (Thornthwaite, 1948). The model implementation (Figure 1) consists of a climate component, science components (e.g., length of daylight, evapotranspiration, snow/rain accounting, soil water balance, and runoff), a component for output, and a component for model execution control. Thornthwaite was chosen since it has a typical structure of a hydrological simulation model and its size and complexity are manageable for this study. Thornthwaite was implemented in five modeling frameworks utilizing three languages. All model implementations were coded to produce identical numeric output. Programming language specific formatting functions were not used. The FORTRAN 95 implementation consisted of 244 lines of code. The model implementations only utilized framework support for component aggregation and interaction/communication. Time stepping was driven by the data input file, and not accomplished by using framework specific functionality. This initial evaluation of environmental frameworks only made use of basic fundamental aspects of the framework.

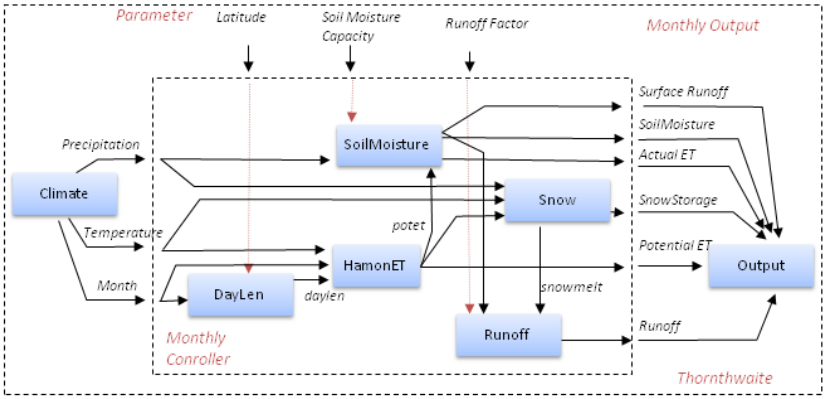

Figure 1. Schematic of Thornthwaite model components (from McCabe and Markstrom, 2007).

The Earth Sciences Modeling Framework 3.1.1 (ESMF) was used to implement a C and FORTRAN version of Thornthwaite. ESMF is open source software developed by the National Center for Atmospheric Research (NCAR) for building climate, numerical weather prediction, data assimilation, and other Earth science software applications (Collins, 2005). The Common Component Architecture 0.6.6 (CCA) was used to implement a Java version of the model. The CCA is developed by the members of the Common Component Architecture Form, and is a standard component architecture for high performance computing. Features of the CCA include multi-language, multi-dimensional arrays, and a variety of network data transports not typically suited for wide area networks (Armstrong, 1999). The Open Modeling Interface 1.4 (OpenMI) was used to implement a Java version of the model. OpenMI is sponsored by the European Commission LIFE Environment program and is a software component interface definition for developing models in the water domain. OpenMI supports rescaling temporal/spatial data so that components operating on data with different geometries can interoperate seamlessly (Blind, 2005). The Object Modeling System (OMS) version 2.2 and prototype version 3.0 were used to provide two Java implementations of the model. OMS developed by the USDA – Agricultural Research Service in cooperation with Colorado State University facilitates component-oriented simulation model development in Java, C/C++ and FORTRAN. OMS provides an integrated development environment (IDE) with numerous tools supporting data retrieval, GIS, graphical visualization, statistical analysis and model calibration (Ahuja et al., 2005). Static analysis tools that supported analysis of FORTRAN, C/C++, and Java were used to analyze the model implementations. SLOCCOUNT was used to count lines of code. Understand 2.0 Analyst was used to collect software metrics. Function and data type usage reports produced by Understand 2.0 were parsed using a custom program which generated data for the FDT and FF usage measurements. FDLOC were determined manually by counting lines of code.

4. EXPERIMENTAL RESULTS

The size of the model implementations in lines of code (LOC) are shown in Table 2. We observed a five-fold variation in model size (from a low of 295 LOC for OMS 3.0 to a high of 1635 LOC for CCA). The Common Component Architecture framework (CCA) produces a large amount of automatically generated code. To normalize the CCA model size for this study we counted only the Java files which were programmer edited during model implementation.

Table 2. Thornthwaite size and complexity

Model Implementation	Total LOC	Average CC/method	Total CC
FORTRAN	244	3.33	40
OMS 3.0	295	2.38	31
C++	405	2.41	41
OMS 2.2	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	880	1.61	116
CCA 0.6.6	1635	2.25	276

Complexity measures for the models are shown in Table 2. Cyclomatic complexity (CC) should reflect the computational complexity of the model. In this study, the model was fixed so the only variation in complexity should be attributed to the use of different frameworks. We observed a two-fold variation in average CC/method and a seven-fold variation in total cyclomatic complexity.

Coupling measures for the models are shown in Table 3. For object-oriented coupling we observed a two-fold variation in total fan-in (efferent coupling) and a three-fold variation in total fan-out (afferent) coupling. For coupling between object classes (CBO) only three systems could be measured since CBO was not measurable for the C and FORTRAN implementations. Additionally, OMS 2.2 uses XML configuration files to specify component interaction resulting in zero measurable CBO.

Table 3. Thornthwaite coupling measures

Model Implementation	Total Fan-In	Total Fan-Out	Average CBO/class
FORTRAN	n/a	n/a	n/a
OMS 3.0	116	70	.89
C++	75	115	.89
OMS 2.2	116	70	0
ESMF 3.1.1 C	100	155	n/a
ESMF 3.1.1 FORTRAN	n/a	n/a	n/a
OpenMI 1.4	126	177	1.1
CCA 0.6.6	195	215	0

For the framework-based model implementations of Thornthwaite measurements for total size, complexity and coupling appeared to move together. 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 $r = .92$ ($df = 3$, $p < 0.05$), and total cyclomatic complexity with total fan-in had a correlation coefficient of $r = 0.95$ ($df = 3$, $p < 0.02$).

Table 4. Model invasiveness rankings

Model Implementation	INV raw	INV %	INV density/ KLOC
OMS 3.0	1	1	1
OMS 2.2	2	4	2
ESMF 3.1.1 C	3	3	5
OpenMI 1.4	5	2	6
CCA 0.6.6	6	5	3
ESMF 3.1.1 FORTRAN	4	6	4

Of the frameworks evaluated in the study, all with the exception of the OMS 3.0 prototype can be classified as heavyweight frameworks as described in Section 1. They all generally provided special implementations of native language data types and numerous API functions which required calling to implement component definition and communication. Using the invasiveness measures proposed in Section 2, an ordinal ranking of the invasiveness of the model implementations is shown in Table 4. The three overall invasiveness metrics combine the measures using arbitrary weights for FDLOC, FDT, and FF measures. Table 5 shows the results of the individual invasiveness measures. FDLOC, FDT-used, FF-used, and INV raw were shown to correlate with model size ($r > 0.811$, $df = 4$); however, none of the percentage or density invasiveness measures correlated with size. For complexity, three raw invasiveness measures (FDLOC, FDT-used, and FF-used) were shown to correlate with total cyclomatic complexity ($r > 0.811$, $df = 4$). The INV raw correlation was not significant. ($r = 0.72$). A correlation existed between FF-used/KLOC and average method cyclomatic complexity. However, when looking at the correlation coefficients for other measures with average CC/method they 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 and total fan-out correlated significantly with INV raw, FDLOC, FDT-used, and also %FF-used (fan-in only) ($df=3$, $p < 0.05$).

Table 5. Framework invasiveness measurements

Implementation	INV Raw	FDLOC	FDT-used	FDT-uses	FF-used	FF-uses
OMS 3.0	-1.38	44	1	1	8	21
OMS 2.2	-.35	147	5	72	7	33
ESMF 3.1.1 C	.05	178	10	122	13	77
ESMF 3.1.1 FORTRAN	.25	280	3	109	11	148
OpenMI 1.4	.49	338	8	73	20	280
CCA 0.6.6	.92	533	15	135	48	215
Implementation	INV %	% FDLOC	% FDT-used	% FDT-uses	% FF-used	% FF-uses
OMS 3.0	-.61	14.84%	4.67%	1.35%	26.67%	40.38%
OpenMI 1.4	-.08	38.41%	23.53%	32.30%	37.74%	79.10%
ESMF 3.1.1 C	-.01	30.85%	30.30%	49.59%	46.43%	76.24%
OMS 2.2	.14	32.67%	41.67%	64.29%	50.00%	73.33%
CCA 0.6.6	.18	32.60%	46.88%	49.82%	70.59%	69.58%
ESMF 3.1.1 FORTRAN	.35	41.42%	27.27%	51.90%	78.57%	96.10%
Implementation	INV density/ KLOC	FDLOC/ KLOC	FDT-used/ KLOC	FDT-uses/ KLOC	FF-used/ KLOC	FF-uses/ KLOC
OMS 3.0	-1.06	148	3.39	3.39	27.12	71.19
OMS 2.2	-.15	327	11.11	160.00	15.56	73.33
CCA 0.6.6	.13	326	9.17	82.57	29.36	131.50
ESMF 3.1.1 FORTRAN	.21	414	4.39	159.59	16.11	216.69
ESMF 3.1.1 C	.41	309	17.15	209.26	22.30	132.08
OpenMI 1.4	.46	384	9.09	82.95	22.73	318.18

5. SOFTWARE QUALITY IMPLICATIONS

One motivation for measuring framework to application invasiveness is to understand how framework design impacts software quality attributes of applications implemented in the framework. For this initial study we did not have resources to directly investigate relationships between software quality and invasiveness. Chidamber and Kemerer (1994) provide a set of object oriented software metrics which have been shown to be useful as indirect measures of software quality (Briand, 2000; Basil, 1996). The Chidamber and Kemerer (1994) metrics are all inversely related to software quality. Higher values generally indicate that the code is more complex and difficult to work with. The metrics include weighted methods per class (WMC), coupling between object classes (CBO), response for a class (RFC), lack of cohesion between methods (LCOM), depth of inheritance tree (DIT), and number of children (NOC). We can use this approach to indirectly evaluate application quality; however, our investigation was limited in that only four model implementations were object-oriented. We found that INV % was correlated with total WMC and total RFC suggesting that

INV % may reflect complexity of the application implementation. A larger study is necessary to further investigate relationships between invasiveness and indirect software quality measures.

6. SUMMARY

This paper presents a unique comparison study where a simple hydrology model, Thornthwaite, was implemented across a set of environmental modeling frameworks. In total, eight versions of the Thornthwaite model were implemented of which six were framework based. The implementations exercised component definition and communication/interoperability features of each framework and each produced identical output for a fixed input data file. Framework invasiveness measures were developed to quantify three dimensions: framework data type usage, framework function usage, and framework dependent code quantification. The measures were used to rank invasiveness of the model implementations. The resulting implementations varied five-fold in size from a low of 295 LOC for OMS 3.0 and a high of 1635 LOC for CCA. Total cyclomatic complexity between model implementations varied up to seven-fold, and coupling between model components varied up to three-fold. We observed a positive correlation between several invasiveness raw count measures and total size (LOC), total cyclomatic complexity, and total afferent/efferent coupling. We found that the total size, complexity, and coupling of the model implementations were positively correlated. To investigate software quality implications of invasiveness, we investigated relationships between the Chidamber and Kemerer (1994) object oriented metrics and the invasiveness measures but limited data prevented a thorough evaluation. Future framework invasiveness should examine additional models with different complexity levels from different domains to better evaluate the utility of the framework invasiveness measures as well as significant relationships between framework invasiveness and software quality.

REFERENCES

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

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

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

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

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

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

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

Dig, D., and Johnson, R. (2006), How do APIs evolve? A story of refactoring. *Journal of Software Maintenance and Evolution: Research and Practice*, 18, 83-107.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995), Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

McCabe, G.J., and Markstrom, S.L. (2007), A monthly water-balance model driven by a graphical user interface. USGS Open-File report 2007-1088, 6 pp.

Richardson, C. (2006), POJOs in Action: Developing Enterprise Applications with Lightweight Frameworks. Manning Publications Co., Greenwich, CT.

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

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