# Development of an object oriented lattice QCD code "Bridge++"

**S Ueda[1], S Aoki[2], T Aoyama[3], K Kanaya[4], H Matsufuru[5], S Motoki[6], Y Namekawa[7], H Nemura[7], Y Taniguchi[4] and N Ukita[7]**

[1] Theory Center, IPNS, High Energy Accelerator Research Organization (KEK), Tsukuba 305-0810, Japan
[2] Yukawa Institute for Theoretical Physics, Kyoto University, Kyoto 606-8502, Japan
[3] Kobayashi-Maskawa Institute for the Origin of Particles and the Universe (KMI), Nagoya University, Nagoya 464-8602, Japan
[4] Graduate School of Pure and Applied Sciences, University of Tsukuba, Tsukuba 305-8571, Japan
[5] Computing Research Center, High Energy Accelerator Research Organization (KEK), Tsukuba 305-0801, Japan
[6] University of Aizu, Aizu-Wakamatsu 965-8580, Japan
[7] Center for Computational Sciences, University of Tsukuba, Tsukuba 305-8577, Japan

E-mail: sueda@post.kek.jp

**Abstract.** We are developing a new lattice QCD code set "Bridge++" aiming at extensible, readable, and portable workbench for QCD simulations, while keeping a high performance at the same time. Bridge++ covers conventional lattice actions and numerical algorithms. The code set is constructed in C++ with an object oriented programming. In this paper we describe fundamental ingredients of the code and the current status of development.

## 1. Introduction

A perturbative approach to QCD succeeds in the high energy physics such as the deep inelastic scattering. On the other hand, it does not apply to the low energy phenomena such as the quark confinement and the light hadron spectrum. Some nonperturbative methods are necessary. Lattice QCD is a SU(3) gauge theory defined on 4-dimensional Euclidean lattice, which provides a nonperturbative framework of the calculation [1]. The path integral quantization enables Monte Carlo simulations.

Rapid increase of the processor performance and the integration technology of the large-scale cluster system enables us to apply the lattice simulation to wide range of research area: matrix elements to test the standard model in LHC and B factory experiments, the phase diagram and the nature of QCD at finite temperature and density, the force among nucleons, and so on. The lattice framework is also applied to gauge theories other than QCD, such as a technicolor theory to explain the mechanism of the Higgs sector of the standard model.

While numerical lattice simulations have become an indispensable tool for theoretical analysis, the programming technique has become more and more involved. The hybrid parallelization with MPI and OpenMP is inevitable for the current large-scale system such as K-Computer and Blue Gene/Q. GPGPUs require an additional development of the code with CUDA or OpenCL. The

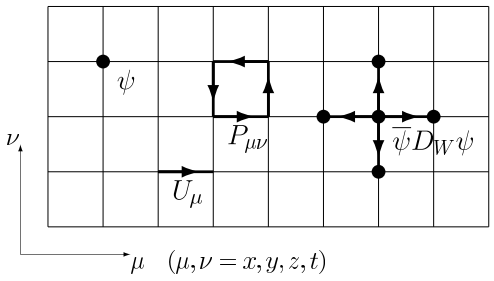


**Figure 1.** Gluon field $U_\mu$ and quark $\psi$ on the lattice. The plaquette $P_{\mu\nu}$ as well as an interaction of fermion operator $\bar{\psi}D_W\psi$ are also given.

research groups have to develop new codes for updated machines. It makes these codes hard to be understood for beginners. In particular, it is a serious problem that new ideas or new physical quantities cannot be tested quickly.

In view of this programming situation, we started a project to develop a new common code set "Bridge++" for lattice QCD simulations. We adopt C++ language to make use of the object-oriented design. Bridge++ has a great deal of readability while keeping sufficient performance for frontier works. The machine dependent part of the code is hidden as much as possible. The first version of the code was released in July 2012 under the GNU General Public License. It is actively being developed to extend functions, brush up its design and the implementation, improve the performance, and provide more documents. The latest version is 1.1.1. The code is available from our website [2].

This paper is organized as follows. In the next section, the core constitution of the lattice QCD simulation is briefly summarized. In Sec. 3, the policy of development of Bridge++ project is described. Section 4 summarizes our current status of the project and future prospects.

## 2. Lattice QCD simulations

### *2.1. Principle of lattice QCD simulations*

K.G. Wilson proposed to formulate QCD on a discretized Euclidean spacetime, namely on a lattice [3]. It enables a nonperturbative analysis of QCD by numerical simulations.

In lattice QCD, the gauge field $A_\mu$ is represented as $3 \times 3$ complex matrices on the link connecting the nearest neighbor sites such that $U_\mu(n) = \exp(igA_\mu(n))$, where $g$ is the strong coupling constant and $n = (n_x, n_y, n_z, n_t)$ is a lattice site (Fig. 1). The number of lattice sites is finite in the numerical simulation by the lattice size $L_\mu$, $n_\mu = 1, \ldots, L_\mu, \mu = x, y, z, t$. The quark field $\psi$ is represented as a complex vector on a lattice site which carries 3 components of color and 4 components of spinor.

Applying the path integral quantization, an expectation value of observable $\mathcal{O}$ is given by a functional integral

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int \mathcal{D}\overline{\psi}\mathcal{D}\psi\mathcal{D}U_\mu \mathcal{O}(\overline{\psi}, \psi, U_\mu) e^{-S_{\rm lat}} = \frac{1}{Z} \int \mathcal{D}U_\mu \mathcal{O}(U_\mu) \det(D) e^{-S_G}, \tag{1}$$

where $Z$ is the partition function and $S_{\rm lat} = S_G + S_F$ with the gauge part $S_G$ and the fermion part $S_F = \bar{\psi}D\psi$ of the lattice QCD action, respectively. The quark field is integrated by hand and arrives at the second equality of Eq. (1). The quark propagator $D^{-1}$ expresses a propagation from a site to another site. Hadron spectrum can be extracted from the hadron correlation functions that are composed of quark propagators.

The expectation value is calculated by generating ensemble of gauge fields with Monte Carlo method under the probability

$$P(U) \propto \det(D[U]) e^{-S_G[U]}, \tag{2}$$

where $U$ represents a gauge configuration $\{U_\mu(x)\}$. Once the ensemble of independent configurations $U^{(i)}$ $(i = 1, \ldots, N)$ are in hand, an expectation value of the physical observable is obtained by

$$\langle \mathcal{O} \rangle = \frac{1}{N} \sum_{i=1}^{N} \mathcal{O}[U^{(i)}]. \tag{3}$$

It is noted that the expectation value $\langle \mathcal{O} \rangle$ has not only the statistical error but also several systematic errors. Numerical simulations are inevitably performed at finite lattice spacing and finite volume. It is imperative to control these errors for a precise study.

*2.2. Lattice QCD actions*

We introduce the fundamental form of the gauge and fermion actions. An action on the lattice is constructed under the following principles.

- The lattice action must be invariant under the gauge transformation.
- The lattice action must coincide with the continuum theory in the continuum limit, i.e. the lattice spacing $a \to 0$.
- The lattice action should retain symmetries in the continuum theory as much as possible.

The lattice action is not unique, because there is freedom to add a term which vanishes in the continuum limit. Variety of lattice QCD simulations originate from the type of the lattice actions.

The standard gauge action was given by Wilson, and thus called Wilson (plaquette) action:

$$S_G = -\frac{N_c}{g^2} \sum_n \sum_{\mu \neq \nu} P_{\mu\nu}(n), \qquad P_{\mu\nu}(n) = \frac{1}{N_c} \mathrm{tr} \left[ U_\mu(n) U_\nu(n + \hat{\mu}) U_\mu^\dagger(n + \hat{\nu}) U_\nu^\dagger(n) \right], \tag{4}$$

where $N_c = 3$ is the number of color, $n$ is a site, $\hat{\mu}$ is a unit vector in $\mu$-direction. $P_{\mu\nu}(n)$ is the smallest closed loop made of link variables called plaquette. It corresponds to the field strength. This action agrees with the continuum gauge action up to $O(a^2)$ corrections. By adding closed loops, such as rectangular loops, improved gauge action is constructed so as to reduce the $O(a^2)$ effects.

The lattice fermion action is more complicated. A naive discretization of the continuum fermion action results in so-called doubling problem: the propagator possesses undesirable additional poles called doublers corresponding to particles at the edges of the Brillouin zone. There are other ways to circumvent the doubling problem. It provides a type of fermion actions. The simplest idea is to add a second derivative term to eliminate doublers. This leads to the Wilson fermion action constructed from the Wilson fermion operator $D_W$ which connects a site $m$ to $n$ such that

$$D_W(m, n) = \delta_{m,n} - \kappa \sum_{\mu=1}^{4} \left[ (1 - \gamma_\mu) U_\mu(m) \delta_{m+\hat{\mu},n} + (1 + \gamma_\mu) U_\mu^\dagger(m - \hat{\mu}) \delta_{m-\hat{\mu},n} \right]. \tag{5}$$

$\gamma_\mu$ is a $4 \times 4$ matrix acting on the spinor space, and $\kappa$ is a parameter related to the quark mass $m_q$ through $\kappa = 1/(2m_q + 8)$. It should be noticed that the second derivative term violates the chiral symmetry on the lattice, though it vanishes in the continuum limit. It has recently become popular to adopt other fermion operators which respects the chiral symmetry on the lattice at the high numerical cost, such as the domain-wall and overlap operators.

*2.3. Generating gauge configurations*

The simplest hybrid Monte Carlo (HMC) algorithm is explained. HMC is a standard algorithm to generate gauge field configuration from $U^{(i)}$ to $U^{(i+1)}$.

Since $\det D$ is hard to be evaluated, we represent it as an integration of boson fields, called the pseudo-fermion fields. For two flavors of degenerate quark masses, $\det(D^\dagger D)$ is to be evaluated as

$$Z = \int \mathcal{D}U \det(D^\dagger D) e^{-S_G} = \int \mathcal{D}U \mathcal{D}\phi^\dagger \mathcal{D}\phi e^{-S_G - \phi^\dagger (D^\dagger D)^{-1} \phi} \tag{6}$$

We need to integrate the link variable $U$ and pseudo-fermion $\phi$, whose action contains an inverse of the fermion operators, $(D^\dagger D)^{-1}$.

The new configuration is brought by evolutions of $U$ and its conjugate momentum $P$. The Hamiltonian of $P$ and $U$ governs the evolution with respect to the fictitious time that is independent from the original temporal coordinate. During the evolution of $P$ and $U$, $\phi$ is kept fixed as external field. $P$ and $\phi$ are given randomly at the beginning of the evolution equation. In solving the evolution equations numerically, a finite step size provides a systematic error. This error can be compensated by a Metropolis test of the difference of Hamiltonian at the beginning and the end of the evolution.

To summarize, the gauge configuration is generated in the following steps.

- Generate the pseudo-fermion $\phi$ under the probability $P \propto \exp[-\phi^\dagger (D^\dagger D)^{-1} \phi]$, this can be realized by Gaussian random number $\xi$, under the probability $P = \exp(-\xi^\dagger \xi)$, and $\phi = D^{-1}\xi$.
- Set the conjugate momentum $P$ with Gaussian probability distribution.
- Solve the evolution equation for $U$ and $P$ with respect to the time step. To evaluate the force of $P$, one needs to solve a linear equation for a large sparse matrix, $x = (D^\dagger D)^{-1}\phi$.
- After getting a candidate of new configuration $\{P', U'\}$, calculate the deference of Hamiltonian $\Delta H = H[U', P', \phi] - H[U, P, \phi]$ and accept $U'$ as new configuration with the probability $\max\{1, e^{-\Delta H}\}$. If rejected, the old configuration is adopted as the new one.

Using the gauge configurations, expectation values of observables can be computed according to Eq. (3).

## 3. Lattice QCD code set Bridge++

*3.1. Bridge++ project*

Bridge++ project starts on the following background. There are many lattice QCD codes, such as MILC code (in C, USA) [4], CPS++ (C++, USA) [5], Chroma (C++, Europe) [6], and Lattice QCD Toolkit (Fortran, Japan) [7]. We have realized, however, it is problematic that there has been no genuine code set other than Lattice QCD Toolkit in Japan, although Japan is one of the centers of lattice QCD simulations. It is desirable to possess a code set under our management, because it can instantly reflect a feedback of modifications, machine specific tunings, and other improvements. Another reason is that it has no black box. We can confirm and understand the code genuinely. In addition, a new code does not have a phantasm from the development history, which sometimes discourages a beginner.

For these reasons, we have started a development of our new code set, named Bridge++, in 2009. We aimed that this general-purpose code set should have the followings features:

- Readability: the code structure is transparent so as to be understandable even for beginners.
- Extensibility: the code is easy to be modified for testing new ideas.
- Portability: the code runs not only on laptop PC but also on supercomputers.
- High-performance: the code has a high performance enough for productive research.

To achieve these goals, we adopt an object oriented design in C++ programming language. MPI is used for distributed memory machines. In Bridge++ code set, we use several simple idioms repeatedly. Such repetition helps beginners to understand the idioms and the code structure. The code is managed with a repository and bug tracking system.

In July 2012, the first version of Bridge++ was released. It contains fundamental lattice actions, linear algebraic algorithms, and various physical quantities. The documents are compiled on wiki as well as TeX based reports. We also provide a HTML document generated by doxygen [9] from the comments in the source file.

### *3.2. Object-oriented programing*

Object-oriented programing (OOP) is employed in Bridge++. It is based on "Objects", which are sets of data fields and methods. A problem is solved using an interaction between objects. Data field is a characteristic variable of the object, called a member data or a member variable. Method is a function that defines a behavior of the object.

OOP is characterized by the following properties.

- Encapsulation: data is handled through the interface, that defines how to use the object.
- Inheritance: object is expandable by adding new functions.
- Polymorphism: objects with the same kind of behavior can be handled through the same interface.

These properties are mechanisms to maximize reusability of the code. Using OOP, an interface is separated from details of implementation, and localize the latter which is sometimes specific to the architecture due to optimization. The polymorphism allows us to implement an algorithm with a set of objects that have the common interface.

There is a compilation of wisdom to make use of these virtues of OOP. An example is so-called design patterns [10, 11]. The design pattern is a kind of programing idiom that frequently appears as a good solution to certain kind of problems. Their efficiency has been realized after the GoF's publication [10] that classified such idioms into 23 design patterns. The design pattern enables us to use the benefit of OOP easily, as well as to make our code transparent.

In the following, we introduce two design patterns and show how they are applied to our problems.

### *3.3. Bridge pattern and linear equation solver*

Bridge pattern is a design pattern which decouples an interface from its implementation so that they can switch independently. Bridge++ employs this bridge pattern. It has advantages for readability and further developments.

A typical example of the bridge pattern is an implementation of a solver algorithm and a fermion operator. In lattice QCD simulation, the most time consuming part is solving a linear equation for the fermion operator. An example is the Wilson fermion operator in Eq. 5, which contains couplings up to the nearest neighbors, and thus is a large sparse complex matrix with a rank of $3 \times 4 \times N_x \times N_y \times N_z \times N_t$. The propagator is the inverse of the quark operator, $S_q(m,n) = D^{-1}(m,n)$. It is obtained by solving a linear equation

$$\sum_l D(m,l) S_q(l,n) = \delta_{m,n}, \tag{7}$$

where color and spin indexes are omitted. There are numbers of fermion operators and solver algorithms. The best solver algorithm depends on a type of the fermion operator. The solver algorithm should be implemented so as to be applied to any kind of fermion operators, and simultaneously solver algorithms must be changed easily.

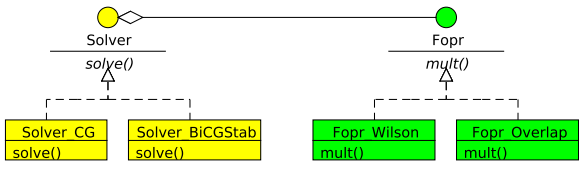


**Figure 2.** Class diagram of the Bridge pattern applied to a linear solvers.

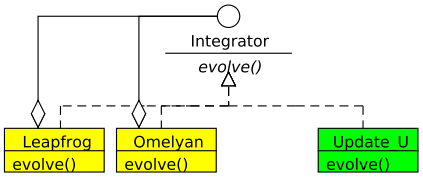


**Figure 3.** Class diagram of the Composite pattern applied to the HMC integrators.

Figure 2 shows the class diagram of the Bridge pattern based on UML (Unified Modeling Language). The interface of the fermion operator is `Fopr`. `mult()` is a function that applies the fermion operator to a given vector and returns the resultant vector. The practical implementation of `mult()` is given in a subclass of `Fopr`, such as `Fopr_Wilson` for the Wilson fermion and `Fopr_Overlap` for the overlap fermion. Similarly the base class `Solver` defines the interface of the solver algorithms. `solve()` function returns the solution of a linear equation for a given source vector. Again the practical implementation is given in subclasses, `Solver_CG` and `Solver_BiCGStab`. Bridge++ users can select any combination such as `Fopr_Wilson` with `Solver_CG`.

### *3.4. Composite pattern and HMC integrator*

Composite pattern is a design pattern that treats a group of objects as a single instance of an object. It is convenient to implement a tree structure or nested objects. In Bridge++, it is used in the HMC algorithm.

HMC evolves $\{P, U\}$ according to the Hamiltonian equation discretized by a leapfrog integrator,

$$\begin{pmatrix} P(\tau) \\ U(\tau) \end{pmatrix} = V(\tau) \begin{pmatrix} P(0) \\ U(0) \end{pmatrix}, \qquad V(\tau) = \left[ V_P\left(\frac{\Delta\tau}{2}\right) V_U(\Delta\tau) V_P\left(\frac{\Delta\tau}{2}\right) \right]^N, \tag{8}$$

where $N$ is the total number of the leapfrog steps and $\Delta\tau = \tau/N$. $V$ describes the evolution,

$$V_P(\Delta\tau) \cdot P(\tau) = P(\tau) + \Delta\tau F(\tau), \qquad V_U(\Delta\tau) \cdot U(\tau) = \exp(iP\Delta\tau) U(\tau). \tag{9}$$

$F(\tau) = \delta S_{\rm lat}/\delta U$ is the force of the conjugate momentum.

Instead of the simple integrator, multi-time step integrator can be adopted. It varies the number of time steps for the action terms, according to the contribution to the force. The integrator is

$$V = \left[ V_P^{(F)}\left(\frac{\Delta\tau}{2}\right) V_1(\Delta\tau) V_P^{(F)}\left(\frac{\Delta\tau}{2}\right) \right]^N, \quad V_1 = \left[ V_P^{(G)}\left(\frac{\Delta\tau}{2N_1}\right) V_U(\Delta\tau/N_1) V_P^{(G)}\left(\frac{\Delta\tau}{2N_1}\right) \right]^{N_1}, \tag{10}$$

where $V_P^{(G)}$ and $V_P^{(F)}$ are the integrator of $P$ with the gauge and fermion forces, respectively. Adjusting $N_1$ by monitoring the size of the forces enables us to reduce the step size error, keeping the total numerical cost fixed. In addition, the leapfrog integrator can be replaced with an improved integrator, such as Omelyan integrator, which reduces the finite step size corrections.

Figure 3 shows the class diagram of the Composite pattern applied to the integrator. The base class of the integrator is defined as `Integrator`, which has a virtual method `evolve()`. Its subclass `Update_U` plays the role of $V_U$. The evolution operators $V$ and $V_1$ are implemented by `Leapfrog` or `Omelyan`. An object of these classes may have other objects of subclasses of `Integrator`. This relation is represented in Figure 3 as lines with diamond. In analogy to the file system, $V$ play a role of the directory, and $V_P$ and $V_U$ are files in the directory.

**Table 1.** Features and our development status of the fermion actions. In the last column, 'public' means that is included in the public version and 'testing' has already been implemented and being confirmed.

| Action | chiral symmetry | numerical cost | Status |
|---|---|---|---|
| Wilson | violated | middle | public |
| Clover | violated | middle | public |
| Twisted mass | violated | middle | testing |
| Staggered | partially remains | low | testing |
| Domain-wall | almost exact | high | testing |
| Overlap | exact | very high | testing |

### *3.5. Parallelization*

For a current computational environment, parallelization is inevitable. Large-scale computers consist of nodes which have own memory devices and communicate with each other via high speed network. In addition, each node has many processor cores sharing memory.

Bridge++ works on one node as well as parallel nodes. In Bridge++, we adopt MPI (Message Passing Interface) for a distributed memory parallelization, i.e. for parallel nodes. An inter-node communication, such as broadcast and one-to-one data transfer, is performed through the functions of the Communicator class, which wraps API functions of MPI. If a machine-specific efficient library is available, the implementation of Communicator class is replaced with the one that uses the library. For an environment without MPI, so-called stub implementation of Communicator class is provided.

It is noted that a shared memory parallelization of Bridge++ is now under development. Hybrid parallelization with MPI and multi-threading has been considered. Two multi-thread libraries, Pthread and OpenMP, are compared based on a working implementation of the Wilson fermion operator. While Pthread can accomplish better performance, it is not easy to apply it to the entire code set. OpenMP has been selected as a primary method to multi-threading and performing performance tuning toward incorporating in the next release.

### *3.6. I/O format*

In Bridge++, simulation parameters are given by ASCII files in YAML format. The parameters are held in `Parameter` object. It passes the parameters to each object. Extraction of values from files is handled by a parameter manager class. For the I/O of field objects, several formats are available including ILDG standard format for the gauge configuration. ILDG (International Lattice Data Grid) is an activity to promote sharing configuration data and standardizing data format and description of metadata [8]. The standard output is classified with a verbose level. At present 4 levels are prepared and the output level is selected for each object, such as a linear solver or an observable. There is a extra mode to generate a message with pragma for ILDG metadata generation.

## 4. Current status and future prospects

The current public version contains major algorithms and observables, though the fermion actions are limited to Wilson and clover fermions. Other fermion actions have been implemented and now being tested. Our status is summarized in Table 1, together with the features of each action. These fermion action can be improved by smearing link variable. The smearing has been implemented in Bridge++.

For linear algorithms, major iterative algorithms, such as CG, BiCGStab, GMRES, are available. Multi-shift solver with CG algorithm and eigensolver with Implicitly restarted Lanczos algorithm are also ready.

The status of performance is as follows. As a typical example, we quote the sustained performance for the HMC with clover fermions. On Hitachi SR16000, the rate to the peak performance is about 5% on 1 node (32 cores). On Blue Gene/Q, the public version runs with 2-3% on 32-nodes, while the latest code under development has improved the most time consuming solver part to more than 10%.

Toward the next public release, we are now incorporating the multi-threading to Bridge++. Performance tuning is now in progress. Active investigation to make use of GPGPU is also underway.

## Acknowledgments

In addition to the authors of this paper, this project has been contributed by many of our colleagues, as listed in a document enclosed in the source code. This project is supported by H20 Grant-in-Aid for Scientific Research on Innovative Areas 'Research on the Emergence of Hierarchical Structure of Matter by Bridging Particle, Nuclear and Astrophysics in Computational Science', Joint Institute for Computational Fundamental Science and HPCI Strategic Program Field 5 'The origin of matter and the universe'. The code was developed and tested on Hitachi SR16000 and IBM System Blue Gene/Q at KEK under a support of its Large-scale Simulation Program (No.12/13-15), Hitachi SR16000 at YITP in Kyoto University, K-computer at RIKEN Advanced Institute for Computational Science, HA-PACS at University of Tsukuba under a support for its Interdisciplinary Computational Science Program (No.13a-25) and FX10 at University of Tokyo. This work is supported in part by the Grand-in-Aid for Scientific Research of the Japan (Nos.20105005, 24540250, 25400284).

## References

[1] There are many textbooks of lattice QCD simulations, for example, Montvay I and Münster G 1994 *Quantum Fields on a Lattice* (Cambridge: Cambridge Univ. Press); DeGrand T and DeTar C 2006 *Lattice Methods for Quantum Chromodynamics* (Singapore: World Scientific Pub.).
[2] Bridge++ website, `http://bridge.kek.jp/Lattice-code/`.
[3] Wilson K G 1974 Confinement of Quarks *Phys. Rev.* D **10** 2445.
[4] `http://www.physics.utah.edu/~detar/milc/`
[5] `http://qcdoc.phys.columbia.edu/cps.html`
[6] `http://usqcd.jlab.org/usqcd-docs/chroma/`
[7] `http://nio-mon.riise.hiroshima-u.ac.jp/~LTK/LTKf90.html`
[8] `http://www.usqcd.org/ildg/`
[9] `http://www.stack.nl/~dimitri/doxygen/index.html`
[10] Gamma E, Helm R, Johnson R, and Vlissides J 1995 *Design Patterns: Elements of Reusable Object-Oriented Software.* (Boston: Addison-Wesley).
[11] Trott A and Shalloway J R 2004 *Design Patterns Explained: A New Perspective on Object-Oriented Design, 2nd ed.* (Boston: Addison-Wesley Professional)