Advancement in the development of an Open Source Object Oriented BPSt: development methodology

Livio Mazzarella – Politecnico di Milano – livio.mazzarella@polimi.it
Martina Pasini – Politecnico di Milano – martina.pasini@polimi.it

Abstract

In order to promote its readability, modularity and maintainability, a new Object Oriented (OO) tool for the simulation of buildings performance, has been developed in the last years. The first results of a comparative validation done on our tool, following the BESTEST standard, have been published in the 2013 IBPSA International Conference. The chosen development methodology aims to achieve efficient and high quality software development in the field of Building Performance Simulation tools (BPSts) and is based on an Open Source (OS) development approach. Given the selected approach, the contribution of volunteer developers should be encouraged and supported. To effectively support the work of an OS community, key aspects are tasks automation, traceability and communication in the developing phase. The implemented development methodology is then based on: 1) the use of a Software Forge (SF) to promote communication between community members and to help in the management of the software development life-cycle, 2) the use of UML diagrams to describe community-agreed architectural decisions and enforce their implementation into the project, in a way that their implementation can be automatically checked, 3) the ability to group single tests of different modules in one automatic test session of validation, which also simplifies final reporting, 4) the use of inheritance, offered by Object Oriented Programming (OOP), to specialize existing classes which, avoiding rewriting, partially automate code writing. Regarding the quality of the tool, the definition of specific standards for programming, documenting and validating is also important. In particular, the validation phase has to be carried out in a well-documented pool of verifiers, and provided as an integral part of the documentation available to the user.

1. Introduction

Some of the most important available BPSts, such as ESP-r and EnergyPlus, have followed, even if in different ways, the Open Source approach since the beginning. Today, both of them have enlarged the public availability of their source code, by exposing their source code repository on a public web site for developers (GitHub for ESP-r and SourceForge for EnergyPlus). The first fundamental advantage gained by following such an approach is related to the possibility to find errors in a shorter time, as Raymond's thesis states: "given enough eyeballs, all bugs are shallow" (Raymond, 1999).

Another tremendous advantage is the union between users and developers, since "treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging" (Raymond, 1999).

Meanwhile, in the IT field, different tools and methodologies have been created to increase programming efficiency, promote communication among involved actors and improve code readability & browsability. Some of these utilities are even more vital when following an OS approach, given the un-schedulable and disperse nature of its community's members. In fact, OS communities need, more than others, tools to:

- provide an organic structure for all process phases (development, validation, testing, etc.);
- aid organizing communication between different community's members;
- help old and new members to understand the project
- promote developers' efficiency, also through automation.

In order to promote the readability, modularity and maintainability of a BPSt, some years ago we started the development of a new OO tool for the simulation of buildings performance, following the best practice of an OS approach. For practical reasons, until the project reaches a critical minimum size, it will not be really open to the world. Anyhow, the design of the project structure is following the OS approach, in order to be ready when such a minimum size is reached.
In this paper, we describe what tools and methodologies have been applied to the development of our OO BPSt.

2. The software forge

In an OS project, the most important points are source code management, software development support and project promotion. This is usually done through a web application, called "Software Forge" (SF). Among other possibilities, for instance the SourceForge web site, we have decided to directly host on our servers an OS SF, at least for this development phase, to avoid being linked with any provider that might not be available in the future. The chosen SF is Allura, hosted on an Ubuntu server, which provides, among others, the following important features:

- source code management systems (SVN, Git);
- issue tracking;
- threaded discussion forums/mailing list; wikis;
- documentation facilities.

The main goals of our forge are therefore to:

- host the source code revision's control, the software's home page, installer and documentation;
- promote communication among community members;
- provide tools for development, planning and managing.

2.1 Hosting the source code

Commonly, in a SF, different places, such as branches, tags, and trunk, are devoted to host different versions of the current project. In out SF the "branches" are used to test critical changes before incorporating them into the main development branch, located in the "trunk", while the "tags" are used to host stable releases of the project. Every time the repository is changed, through a commit, a message is associated with it to briefly explain the reason for the new commit. Effective Graphical User Interfaces (GUIs) that show differences in the source code between chosen revisions are provided by Allura.

2.2 Promoting communication among actors

An efficient communication between involved actors is essential, not only to avoid misunderstandings in what should be done, but also to improve planning and strengthen the involvement of each member. Consequently, in our project we have tried to identify appropriate places and tools to improve communication:

- with the user (Documentation, Download and Install web pages);
- with the developer (development procedure, support tools identification and description, etc);
- between developers (dashboards to understand who is working on what, what is he/she doing and with which plan, which are the unassigned activities, etc);
- between users (mailing lists and forum to share experiences);
- between users and developers (issue tracking for discovering bugs & performance bottlenecks , dashboards for suggesting innovation opportunities, or localizing shortfalls, links with feedbacks on addressed/solved issues, etc.);
- between building and systems component manufactures and users (promotion of real products in the DB or DLL of the software).

In fact the SF should put as many stakeholders as possible in contact with each other, starting from the users (both at design and operation time), to manufacturers. Linking users, developers, industries, software models and building

performance monitored data is a vital key to improve design from the smaller scale to policy making at the larger scale.

Consequently, when users need something to be changed, they can start a discussion in the SF to understand if the required change is agreed by the community, and, after that, they can submit their request through tickets.

Another important ingredient for the good evolution of the project is user feedback on addressed issues and requests. Every time a request is made, feedback is welcomed to strengthen the motivation and sense of belonging to the community of each member.

On the other hand, manufacturers willing to promote the performances of their products can find in such an environment a third party legitimation. In fact, manufacturers can ask their products to be added to the project in two ways. If they ask to add a new model, representing their system, to the project (in case such a model has not yet been implemented), this new model should pass the validation procedure. Otherwise, they can ask to add a "prefilled class" to the appropriate DB of the project (i.e. the DB used to collect that kind of products). In this case, the user will find in the DB an object with "pre filled" and unable to be modified, which identifies that particular product. In this second case, the manufacturer will be required to perform experimental tests and provide their documentation to the user together with the prefilled class. Maybe, at the beginning, more effort will be required from the manufacturer, but "free publicity" will be gained as a counterpart. In the meantime, ease of use will be gained by the user, who will be allowed to choose from a list of "correctly" precompiled objects, existing in the real market.

2.3 Communicating current and future project's state

Concerning developers, one of the most important parts in the management of the development is effective communication of what has already been done, what should be done, who is doing what and what are their plans when doing it.

The communication concerning the current and future state of the project takes place, for our project, inside the Integrated Development Environment (IDE). This is due to the possibility, provided by the used IDE, which is Visual Studio Ultimate 2013, to use an UML diagram linked with the source code. This opportunity improves readability and browsability of existing code and the communication of software decisions. This part will be discussed in section 3.1: Modelling tools.

Inside the SF milestones and tickets are created and managed. This second kind of objects is meant to list current, past and future activities involved in the development process and contains process information such as Author/Creator, Assignees/Owners, Priority, estimated work-effort or time-to-complete, Dependency/Predecessor, Tested by, Labels, related Milestone, Status, etc.

This variety of information, contained in the SF, should be "captured" and synthetized to community's members to provide all the stakeholders, in an easy-to-consume form, statistics on the life and features of the project.

Thus, not only work progress, but also statistics about user feedbacks, model validation results, etc., should be reported in such "dashboards". Unfortunately, such an enriched dashboard still does not exist. However, it could be easily implemented in the future if the validation procedure and the software structure allows it.

3. Promoting Programming Efficiency

Programming efficiency is promoted by:

- communication;
- an appropriate software structure (which allow code reuse);
- the implementation of automatic routines/activities.

3.1 Modelling tools

As previously said, the communication of what has been done, which models are waiting to be implemented and which architectural choices have been taken, is of great importance to promote development efficiency.

This kind of communication is implemented

through UML diagrams and has been grouped in a Modelling Project (linked with the rest of the source code) inside our software solution. As a matter of fact, the selected IDE allows a full coupling of UML diagrams and source code, automatically writing code while visually creating a diagram or automatically generating or modifying diagrams when the code is changed.
The generated UML diagrams help in:

- exploring existing architectures;
- understanding activities' dependency and sequence;
- specifying and enforcing (with layer diagrams) the structure or behavior of a system;
- providing a template that guides in constructing a system;
- documenting decisions, etc.

3.2 Software structure & features

The structure of our software has been conceived to simplify modifications of the source code. The OO paradigm perfectly responds to this aim by encapsulation and inheritance.
Everything is a class with properties that exposes a behavior and hides its implementation. In this way, developers can easily modify and extend the system limiting the effects on other parts and reducing the line of written code (when inheriting by a father class).
Besides, through reflection, the addition of a new class inheriting by a father class, allow the automatic list on this new item in the GUI to the user.
The chosen programming language is C#, which fulfills such requirements and also combines the .NET framework portability with enough high efficiency and a powerful Integrated Developing Environment (Visual Studio).
The .NET Framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It is open source and Microsoft with .NET 2015 is extending it to run on Mac OS platforms and Linux. (Microsoft, 2014a and Microsoft, 2014b).
Before .NET 2015, the Xamarin MONO project, an open source implementation of Microsoft's .NET Framework, based on the ECMA standards for C# and the Common Language Runtime, assured code portability to Linux and Mac OS platforms, even if with some compatibility issues.

3.3 Tasks' Automation

The used IDE also provides tools that help to check that architectural and programming guidelines have been correctly implemented.
This family of tools, addressed with "profiling tools", measures a system's class structure, coupling, complexity, cohesion, memory allocation, CPU use, resource contention, etc.
Applying such automatic testing helps find bottlenecks or low-performing code and can be useful for a first check of codes developed by others.
Together with the automatic check of the "programming quality" of the source code, other useful tools, provided by the IDE, for task automation are snippets and unit tests.
Snippets are code template that can be easily called while programming to be pasted in a context. They have been used, for example, to provide a similar structure to different classes (such as different code's regions for "Constructors", "Private Fields", "Public Methods", "Virtual Methods", "Overridden Methods", etc.).
Unit tests, however, have been used to explicitly declare, given some input, the expected output of a specific part of a program. After their implementation, it is possible to automatically check all the unit tests created inside the project with only one click. This event will trigger the execution of all the tests and the generation of a report showing which tests are met and which are not.
The used IDE also provides tools to scan the percentage of coverage of the code with tests, encouraging a development methodology where code and tests grow in parallel to assure the quality of the software produced in each iteration.

4. Development and Validation (D&V) procedures

The focus of D&V procedures is on simplifying/automating these activities as much as possible.
As procedures should be repeatable, apart from possible bifurcations, they should be translated as much as possible into automatic tasks or templates, as we have seen in the previous paragraph, with snippets, unit test checking, etc.
Even if automation is not strictly possible, having a document that explains the commonly agreed best way to test a module, or a spreadsheet for comparing results, will allow for their search and implementation to be skipped.

4.1 Development procedures

Regarding the development procedures provided in our SF, they consist basically in programming guidelines and explanation about which contribution should be submitted in which context inside the SF or IDE, as previously explained.
The programming guidelines, meant to "assure" that each developer's approach is consistent with that of the others, cover a range of subjects, starting from naming and usage conventions, arriving to performance and security considerations. However, having written guidelines does not guarantee that developers will read and follow those practices.
The desire to automate the process of evaluating code for compliance with these guidelines led to the creation of Code Analysis tools implemented inside the IDEs. These tools are based on rules, also grouped by subjects, ad-hoc defined for the project or already implemented as a result of the numerous years of experience of a specific community of developers. Once enabled and configured, code analysis will be performed at each build and a report will be automatically generated.
Besides the standard for code writing, some standards for the documentation of the code are provided too.
Documentation and code writing should go hand in hand. The documentation provided is located inside the code, through comment and indentation, and is created in three kinds of documents, i.e:

- the User Manual,
- the Engineering Manual
- and the Validation Reports.

In the User Manual there are also sections to explain to users how to interact with developers and become a vital part of the community. In this way unforeseen problems collected by users can be directly forwarded to developers for fast analysis and correction.
Technical information about each implemented model is contained in the Engineering Manual.
The detailed description of Validation Reports will be addressed in the next section.

4.2 Validation procedures

Once a new module has been developed, before it can be included in the "main" project, it should pass the validation process.
To complete this process:

- the developers have to redact the Validation Report for their module and produce all the associated results;
- a figure belonging to the community, like the Editor does for a scientific journal, has to select, according to criteria of impartiality and competence, a shortlist of eligible validators, among community members;
- those community members belonging to the shortlist that will accept that specific task will have to control and legitimize the success of this phase.

The selected validators, as happens in peer review processes, should be unconnected with the developers, should have enough knowledge of the problem addressed by the module to be validated and preferably should be in the number of two per each validation procedure.
Each Validation Report, related to a specific component or part of code, contains a short summary and a detailed description of the module and its validation results. A template for this report is available on the SF, partially based on the template developed for by Nordtest Company (Torp, 2003).
The short summary is meant to "present" the new module. It will contain information to locate the

new module in the project, a synthetic description of the objectives and scope of the module and its developers' and validators' identification.
To locate the module, the namespace in which the module is created (ExtendedMath, Utilities, etc.) should be declared and pictures of class diagram and sequence diagrams, with the new module highlighted, should be provided. As a matter of fact, locating the new component among the others in "space and time/activities" is important because it states what the "nature" of the module is.
Depending on the nature and complexity of the physical problem implemented by a computational model and on our knowledge of its behavior, we can have different terminology and validation procedures.
Following a flow chart, developer are able to identify the types of test they have to perform to assess the "accuracy" of the model.
A physical process modelling requires the incremental identification of a Physical/real Model (PM), described by a more or less simplified Mathematical Model (MM), which can have an analytical or numerical solution, implemented on a Computational Model (CM).
First, the MM should be well posed or stable, i.e. it should admit a unique solution that depends on the continuity on the data (Quarteroni et al., 2007). Otherwise, without doing anything else (MM regularization) we cannot pretend that a numerical method applied to it will solve its pathologies (Quarteroni et al., 2007).
Secondly, the simplifications made to create the MM should be "quantified" as much as possible. One or more parameters should be identified to define a range of applicability of the MM to reach an accuracy that is adequate for the intended use. Thus, we should define the Application Domain (AD) for that model. This phase is sometimes referred to as: MM Qualification and might be seen as a quantitative evaluation of the Consistency of the MM to the PM.
After that, when possible, the mathematical model should be characterized by an "error", due to the uncertainties involved with the experimental observations (measurements) of its input parameters. Technologies with similar performances might have different solution errors, thus, such information may drive choices made during the design of the system. If also a sensitivity analysis is possible on the MM, it might be useful to identify unexpected/wrong high sensitivity to certain physical parameters of the CM.
The MM can have an analytical solution or might need a numerical method (consistent and stable, thus convergent) to be solved. This second case is the most problematic, since going from a continuum to a discrete space, some information is lost and consequently, great care should be devoted to the characterization of discretization's errors.
On the CM a validation campaign should be performed to cover, as much as needed, the AD identified for that model with the Validation Domain (VD) for that CM. Vice versa, we will need to provide supported inferences to allow this possibility.
When trying to validate the solution of a numerical method with "exact" or "pseudo-exact" solutions, before continuing with the comparison, discretization errors should be carefully removed, for example, through Grid Convergence Index analysis.
To evaluate the solution of the CM, we can use:

- "exact" analytical solution -generally available only for very specific Boundary Conditions (BCs)-;
- "pseudo-exact" experimental measures;
- the results of other tools already validated.

Depending on the availability of analytical solutions, or experimental measurements, or other validated software, different precautions should be taken during the validation.
Indeed, an analytical solution for that MM might exist for a limited number of BCs. In this case we will have to take care to collect BCs with "enough" significance. For example validating a model in steady state is a necessary but not sufficient step, since it is not validating its dynamics (the VD will still not span all the AD).
In case exact analytical solutions are not available, pseudo-exact experimental "solutions" can be recorded through monitoring. In this case we can follow, among other methods, three steps described in (Obercampf et al., 2002):

- characterization of the uncertainty of input parameter;
- selection of an ensemble of computations (through statistical methods like Monte Carlo, Latin Hypercube, etc.);
- quantification of the uncertainty of the output.

The validation-metric success criteria chosen in this phase is also extremely important. Mean value and uncertainty range should be compared among computational and experimental data, to gain a good knowledge of the CM's behavior.

More in general, applying these three steps will be a way to obtain the mean value for each simulation's result, instead of a value whose probability is unknown. This information might, again, drive the choice among different technologies during the building's design.

To incrementally increase our certainty about the coverage of the AD by the VD, a good possibility is offered by a continuous interaction with monitoring activities performed at operation time. This ongoing dialogue between prediction and measurement may enhance the credibility of the results of the simulation in the design phase and improve the knowledge of the methods implemented within the instrument, as well as of their range of applicability.

After having validated the CM, performing a sensitivity analysis on it can have the further advantages to:

- control that the same behavior of the MM is shown by the CM, if error propagation and/or sensitivity analysis was possible on the MM, or if experience showed a particular behavior;
- check the sensitivity of the numerical method to the choice of the discretization parameters values and inform the user about that;
- warn the user about which input he/she should choose with more care (e.g. finding a manufacturer that has conducted good tests to characterize the performance of his/her products);
- guide the user on which model should be chosen to calculate specific input of the current model (e.g. systems highly sensible to the Mean Radiant Temperature will require the coupling with the more accurate available model for the calculation of View Factors).

5. Conclusions

In this paper we have tried to summarize the lessons learned during the development of our OO BPSt, aimed at implementing an enriched modularity (Mazzarella et al, 2009) for improving readability, maintainability and easy of validation.

The current developing stage of our BPSt is summarised as follows:

- SF: implemented on an Ubuntu server thanks to the Allura Project, ready and currently used by the internal developing team only;
- code kernel: first parallelized (Mazzarella et al., 2014) beta release -building envelope only- currently under tests (previous tests have been performed on its sequential version and their results have been published in: Mazzarella et al., 2013);
- Engineering Manual: currently under development;
- Programming Standards & Developer Manual: first release;
- Validation Reports: currently under development/revision.

During the development of our BPSt, we have seen the continuous evolution of exciting and promising possibilities offered by today's technologies.

However, homogeneity of achieved results is still needed in order to be able to confront different models with each other and with real systems.

In our opinion, the creation of an environment that tries to help promote communication and cooperation between extremely different words (research, profession, manufacture) and that tries to unite design and operation is the first step to achieve a final goal.

This final goal consists in being able to assert and show with numerous case studies that the whole simulation, together with each implemented model, produces results which are correct enough for the intended use.

6. Acknowledgments

Work done under the "TRIBOULET" project, funded by Regione Lombardia.
Thanks also to Narges Shahmandi Hoonejani, who as contributed to the implementation of our SF.

Nomenclature

Acronyms

AD	Application Domain
BCs	Boundary Conditions
BPSts	Building Performance Simulation tools
CM	Computational Model
DB	Data Base
DLL	Dynamic Link Library
D&V	Development and Validation
GUIs	Graphical User Interfaces
IDE	Integrated Development Environment
MM	Mathematical Model
OO	Object Oriented
OOP	Object Oriented Programming
OS	Open Source
PM	Physical/real Model
SF	Software Forge
UML	Unified Modeling Language
VD	Validation Domain

References

Mazzarella L., Pasini M., 2019. Building energy simulation and object-oriented modelling: review and reflections upon achieved results and further developments. *Conference Proceedings of "IBPSA Building Simulation 2009"*, University of Strathclyde, Glasgow, Scotland, 27th - 30th July, (pp. 638-645).

Mazzarella L., Pasini M., 2013. Development of a new tool for the co-simulation of multiple autonomous object. In "Building Simulation 2013", *Proceedings of BS2013: 13th Conference of International Building Performance Simulation Association*, Etienne Wurtz Ed., Chambéry, France, August 26-28 2013 , (pp. 3794- 3801), ISBN 978–2–7466–6294–0

Mazzarella L., Pasini M. and Shahmandi Hoonejani N, 2014. Challenges, limitations, and success of cloud computing for parallel simulation of multiple scenario and co-simulation. *ASHRAE/IBPSA-USA, Building Simulation Conference*, Atlanta, GA, September 10-12, 2014

Microsoft, 2014a, ".NET Core is Open Source". .NET Framework Blog. Microsoft. Retrieved 12 November 2014. Accessed November 28th 2014. http://blogs.msdn.com/b/dotnet/archive/2014/11/12/net-core-is-open-source.aspx

Microsoft, 2014b, Microsoft takes .NET open source and cross-platform, adds new development capabilities with Visual Studio 2015, .NET 2015 and Visual Studio Online. Accessed November 30th 2014. http://news.microsoft.com/2014/11/12/microsoft-takes-net-open-source-and-cross-platform-adds-new-development-capabilities-with-visual-studio-2015-net-2015-and-visual-studio-online/

Oberkampf, W. L., Trucano, T. G., & Hirsch, C., 2004. Verification, validation, and predictive capability in computational engineering and physics. *Applied Mechanics Reviews, 57(5)*, pp 345-384. doi:10.1115/1.1767847.

Quarteroni, A., Sacco, R., Saleri, F. (2nd Edition), 2007. *Numerical mathematics* (Vol. 37). Springer.

Raymond, E. S., 1999. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly Media. ISBN 1-56592-724-9. Accessed November 28th 2014. http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html

Torp, C.E., 2003. "Method of Software Validation (NT TR 535 - Validation scheme)". Accessed November 28th 2014. http://www.nordtest.info/index.php/technical-reports/item/method-of-software-validation-nt-tr-535-validation-scheme.html