ASHRAE and IBPSA-USA SimBuild 2016
Building Performance Modeling Conference
Salt Lake City, UT
August 8-12, 2016

MODEL LIKE A PROGRAMMER: TOOLS AND TECHNIQUES FOR GREATER PRODUCTIVITY

Michael O'Keefe and Peter Ellis
Big Ladder Software

ABSTRACT

Computer programmers have developed a number of tools and techniques for dealing with source code, change management, and automation. Building energy modelers can benefit from applying many of these same tools and techniques to simulation input/output data and workflows. In this paper, we will present four ideas from the software engineering domain that we have applied to our modeling work: text-based tools for working with input/output data, version control for distributed change management, templates for minimizing repeated content, and scripting for process automation. We discuss these ideas in the context of how they can be applied today in actual building energy modeling projects.

INTRODUCTION

Building energy modeling can often feel like a grueling exercise in the organization and management of large amounts of information. Some of this work is due to the inherent difficulty of the problem: buildings are complex systems with a multitude of details, scaled by the size of the building. This is the "essential complexity" of modeling. The remainder of the work is an unfortunate byproduct of the software tools that we all use to get the job done. This "accidental complexity" (Brooks, 1987) includes the overhead of entering, formatting, connecting, and managing inputs—everything related to constructing a model that is required by the software tools but is not directly related to the problem. While essential complexity cannot be reduced, accidental complexity *can* be addressed to gain greater productivity.

Rocky Mountain Institute convened a "Building Energy Modeling Innovation Summit" in March 2011. Several issues with the current practice of building energy modeling were enumerated at the meeting (Tupper, 2011). Number two in the list was that modelers have "limited time for critical thinking". We believe that one of the main reasons modelers do not have more time for critical thinking is due to the time spent managing the accidental complexity of their modeling projects.

Programmers face similar challenges in their work. Computer programs are complex systems constructed by teams of programmers. They are often very large. They can contain lots of repeated content and patterns. Programmers also have to debug their programs, make frequent changes, and repeat common tasks.

The software development field has created robust solutions to manage this type of accidental complexity. However, in our experience, many of these most basic and fundamental tools, processes, and techniques from the computer science field are largely unknown to many practicing building energy modelers.

In this paper, we would like to introduce building energy modelers to tools and techniques that can be used today to help manage the accidental complexity of building energy modeling. We describe four categories of tools and techniques from the computer science field that can improve a building energy modeler's productivity. We have personally put all of these into use for building energy modeling and found them to greatly enhance our modeling productivity. The four areas are:

- Text manipulation
- Version control
- Templates
- Scripting.

For each area, we provide an overview, a rationale for why building energy modeler's should be aware of the area, and tips for getting started. Next, we provide some general advice on how to adopt these new technologies and tools. We finish with a summary and conclusion.

About Text Files

For programmers, "source code" is the fundamental input data for software development. Source code consists of "text files" that hold programming language syntax that is ultimately compiled into an executable program. A given program could contain one text file, or hundreds of text files. The text files contain all of the instructions that tell the computer what to do. Programmers spend the vast majority of their time creating and editing text files.

Text files, sometimes called "plain text" files, are exclusively composed of sequences of characters or "text". Text is distinguished from other file formats because it is human readable (even if it might be somewhat cryptic), has no hidden embedded formatting, and is the closest thing to a universal file format—many programs and software tools can read/write text.

Some common text files you might recognize include HTML, XML (including gbXML), CSV, *EnergyPlus* input files (IDF), and weather files (EPW).

In contrast, non-text files are typically binary files such as executable programs (EXE), image files (JPG, GIF, PNG, etc.), *Microsoft Office* documents (DOCX, XLSX), most database files, and *DOE-2* weather files (BIN).

Modelers—like programmers—work extensively with text files in many cases. For modelers, our source code consists of the input files, weather files, and other data files that are required by a given modeling tool. Many of these are text files. The majority of simulation engines (including *EnergyPlus*, *DOE-2*, *TRNSYS*, and *ESP-r*) and some user interfaces (including *OpenStudio* and *eQUEST*) read and write a text-based format for their input files. Other user interfaces (such as *DesignBuilder*) can write out a native text-based input file for their respective simulation engines.

Thus, since many simulation engines *are* text-based applications, it makes sense to leverage tools and practices that enhance our productivity with text. Because programmers spend the vast majority of their time creating and editing text files, they have developed countless tools and techniques to maximize their productivity while working with text. We can adopt many of those tools and practices.

Working with text does not mean you must adopt a text-only workflow. Graphical user interfaces (GUIs) do offer many benefits that you should not have to give up. In fact, many of the tools and techniques we recommend should complement the work you do in your favorite GUI tools. For those using a non-text-based modeling program, some of the advice about text manipulation will still apply although you may be limited in its full application.

TOOLS AND TECHNIQUES

Because much of our source code for modeling is made up of text files, our first step is to introduce two key tools for text manipulation: text editors and diff-merge tools.

Text Manipulation: Text Editors

Text editors are a general type of text manipulation tool. As the name implies, text editors allow you to create and edit text files. A proper text editor is your gateway to the world of text files.

Although simple text editors usually come packaged with your operating system (e.g., *Notepad* for Windows), we discourage their use for modeling work. Although these simple programs can edit plain text correctly, they lack many of the capabilities for processing text *effectively*. We should note particularly that word processors such as *Microsoft Word* are **not** true text editors. Although the words on the screen can look like plain text, the file format is binary and stores additional data about formatting and other information.

Full-featured text editors suitable for programming or modeling typically have the following capabilities:

- Syntax highlighting
- Automatic indentation
- Line numbering
- Advanced search and replace (locally and across files)
- File browser
- Support for multiple programming languages
- Macros.

In addition, the functionality of many editors can be customized or extended through plugins, macros, and/or configuration files.

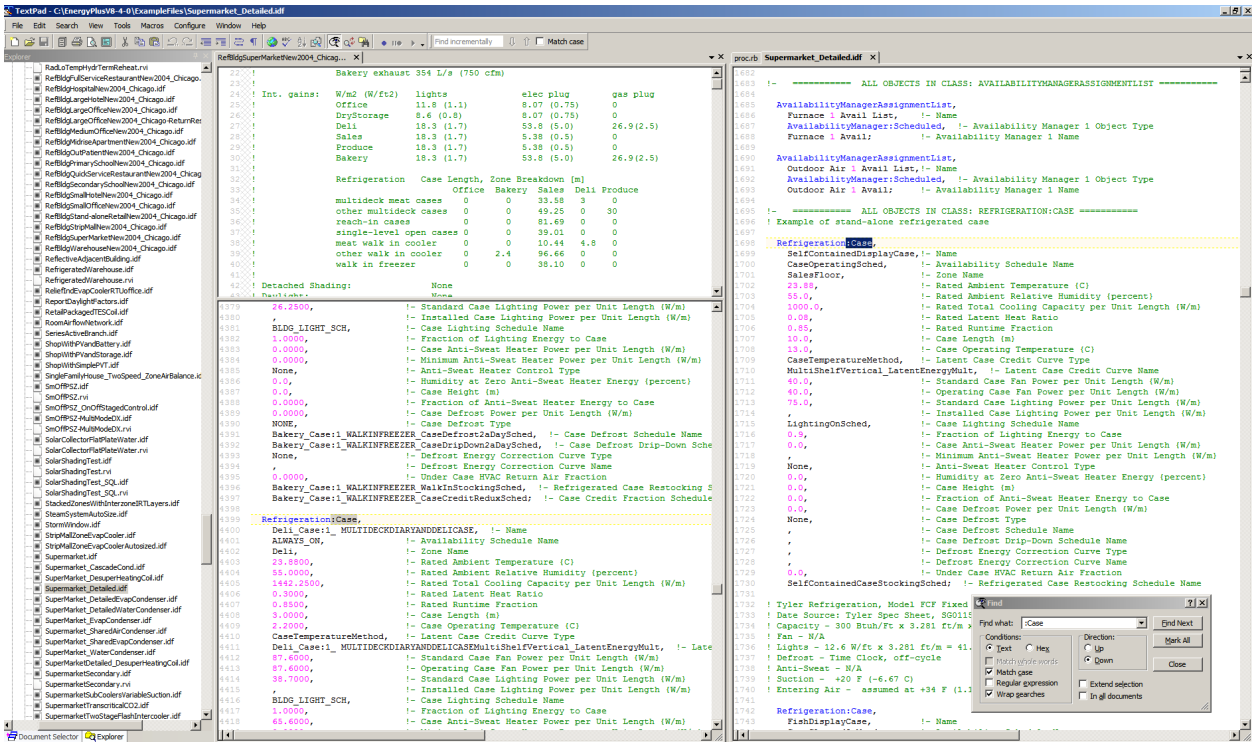


Figure 1. Screenshot of TextPad with Syntax Highlighting for EnergyPlus

Relevance: Many simulation engines and user interfaces read and write text-based input files. A full-featured text editor can increase your productivity when creating and editing input files—even if you mainly rely on a user interface to create your input file.

Syntax highlighting is an especially useful feature that makes it much easier to read input files. By configuring a syntax definition file, you can enable the highlighting of different parts of the input file format so that they are displayed in different colors or fonts. For example, syntax highlighting for *EnergyPlus* could be configured to show the keywords for object types in a blue font while comments are in a green font and numbers are in a magenta font (Figure 1). Depending on the structure of the input language syntax, it may be easier to configure highlighting for some simulation engines than others.

Getting Started: Several popular text editors are available for free or at low cost:

- *Atom* – free and open source; Windows, Mac, and Linux (GitHub Inc., 2016a)
- *Notepad++* – free and open source; Windows only (Ho and Notepad++ Collaborators, 2016)
- *TextPad* – free evaluation; Windows only (Helios Software Solutions, 2016)

Example: Figure 1 shows a screenshot of one of the text editors we use: *TextPad*. Although the text is too small to read, the figure illustrates the major parts of a full-featured text editor. The editor shows a file browser on the far left, two views of different parts of the same input file open in the center, and a second input file open on the right. The bottom right of the editor shows a dialog box for a powerful search feature which can also search across multiple files. The input files also show syntax highlighting for *EnergyPlus*. One might use such a text editor for direct input or to inspect the generated input file from a GUI tool.

Text Manipulation: Diff-Merge Tools

Diff-merge tools are a specialized type of text manipulation program. The primary function of these tools is to provide a visual comparison of the differences ("diffs") between two text files. Typically the user interface shows a split view with the two files compared side-by-side at the same point. Individual lines with differences are highlighted in a distinctive color. Some tools even indicate localized differences within a line (see Figure 2). Most tools have the capability to filter out less important differences such as changes in white spaces only.

The secondary function of diff-merge tools is to "merge", or copy, differences back and forth between the two files. One can select individual lines or blocks

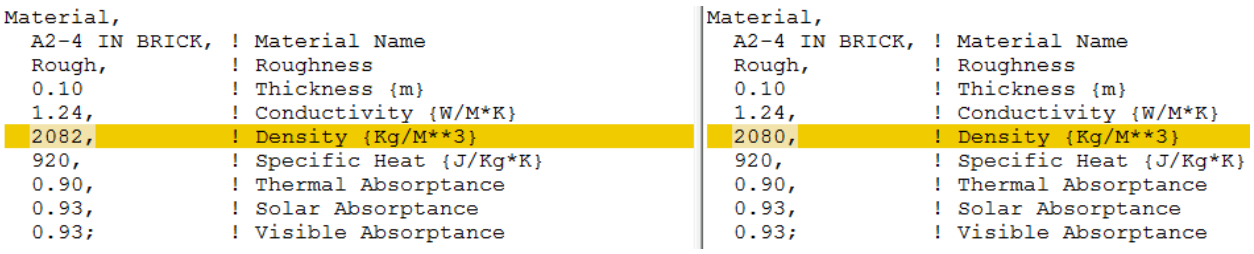


*Figure 2. Diff Between Original and Changed Material Object (*WinMerge*)*

of changes which can be copied to the other file. Differences or changes, in whole or in part, can be quickly merged from one file into the other without the pain of manual copy-and-paste.

Relevance: You can use diff-merge tools to quickly compare two input files, or other modeling-related text files, to detect what differences exist between them. Diff-merge cannot be used effectively with non-text files (with some notable exceptions: some tools can show differences between binary image files, for example).

One case where diff-merge is especially helpful is when debugging a difficult model. During the debugging process it may be necessary to make a number of changes to the input file to identify the correct fix for a problem. Unless you track your changes very carefully, it can be easy to end up with some superfluous changes that you forgot to undo at the end of your debugging. Diff-merge helps you eliminate this headache. Before debugging, make a separate copy of your input file. After debugging, you can "diff" the original file with the fixed file to inspect all of the changes. You can merge only the changes that you want to keep back to the original file while ignoring the other differences. The result is a fixed input file along with the assurance that you know exactly what changed in your model.

Diff-merge is the first step towards "change management".

Getting Started: Numerous free and commercial diff-merge tools are available. In fact, many text editors and user interfaces for version control systems (see next section) have some or all diff-merge capabilities built into them. However, even if you already have diff-merge capabilities available through another tool, we recommend also installing a high-quality, stand-alone diff-merge tool. Several popular tools available for free, or at low cost, are listed below:

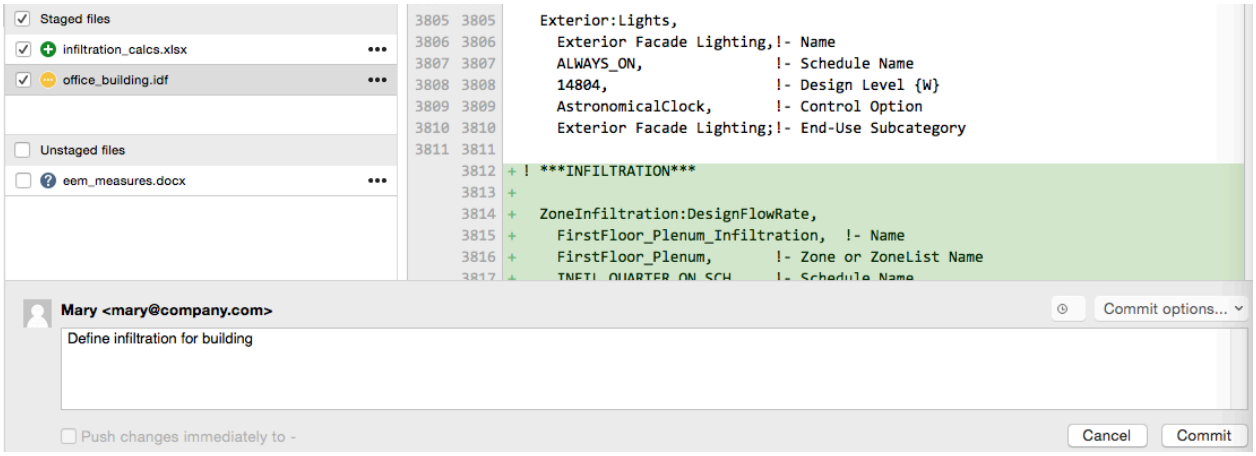
- *WinMerge* – free and open source; Windows only (WinMerge Developer Team, 2013)
- *DiffMerge* – free; Windows, Mac, and Linux (SourceGear, 2016)
- *FileMerge* – distributed with Mac; Mac only
- *Beyond Compare* – commercial; Windows, Mac, and Linux (Scooter Software Inc, 2016).

Example: Diff-merge output formats vary from tool to tool but all are quite similar. Here we present the textual output of the diff between two *EnergyPlus* Material objects where changes have been made to the "density" field. While it may be easy to spot the difference with the naked eye in this contrived example, this is no trivial task for a typical input file consisting of thousands of lines. Finally, we see the raw output of a diff-merge tool in Figure 2—this screenshot was created with the diff-merge tool, *WinMerge*. Although the user interface and features of diff-merge tools differ from program to program, each one provides a way to navigate between differences and allows you to selectively copy changes from one file to the other.

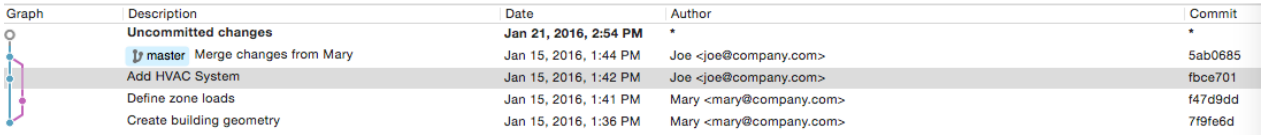
Version Control

Version control is the management of changes made over time to the computer files that comprise a project, including annotations about those changes such as who made them, at what time, and why.

We suspect that most if not all of the practicing energy modelers reading this paper have used some form of version control at one time or another. If you've ever saved duplicate copies of your input file with names like "chicago-hotel-2010-06-07.idf", "chicago-hotel-2010-06-07a.idf", or "chicago-hotel-2010-06-08-revision-from-ben.idf", you've used version control. If you've ever kept a detailed list of changes made to an input file in a Microsoft Word document noting what you changed and why, you've used version control. These are all examples of manual ad-hoc schemes to preserve the history of changes to your computer files. We do this to audit ourselves and to keep a "known good version" of our models available. The problem with these mechanisms is that they are cumbersome, error-prone, non-standard, difficult to recover previous versions in practice, and they don't (typically) make effective use of the other tools we've mentioned such as diff-merge. Furthermore, working with an ad-hoc manual system for tracking changes is tricky enough for one person but it effectively stops parallel team collaboration dead in its tracks because it's just too hard



(a) Commit View



(b) History View

Figure 3. A Graphical Front End for Git's Version Control System

to manage two or more streams of changes from multiple sources.

Automated version control systems arose from the needs of programmers working collaboratively on software projects. The first automated version control systems arose in the 1970s but development and refinement is occurring through present day. These programs automate the management of changes to files along with annotations of those changes with three goals (Sink, 2011):

- To enable collaborative work in parallel
- To provide a means to deal with conflicting changes
- To save a total annotated history of all changes made to the project.

We can adopt these same tools for building energy modeling.

Relevance: Using version control in your modeling and simulation projects may well be the single most important thing you can do to improve productivity.

That is a lofty claim but one that we believe to be true for two main reasons:

- Version control enables distributed, collaborative work
- Version control enables you to track, audit, and "undo" mistakes thanks to the existence of an annotated browsable history of all changes made to your project.

As an anecdotal example, the real power of version control was made clear to us the first time our team collaboratively developed an energy model in *EnergyPlus*. The building was a large multi-story federal office building and the analysis was to evaluate several efficiency options for the building. Three team members were working on the model at the same time from remote locations: one was adding new HVAC systems, another was adding shading devices to all the windows, while another worked on the cases to be studied. By dividing up the work, we were able to develop and track all changes in near real-time, greatly decreasing the calendar time required to deliver our product to the customer.

Getting Started: We recommend getting started with version control by using one of the free open-source systems such as the popular *Git* program (The Software Freedom Conservancy, 2016; Sink, 2011). *Git's* popularity has resulted in a wealth of tutorial and help

information (see, for example Chacon and Straub, 2014). Because version control programs, themselves, are (typically) command-line utilities, we further recommend using one of the free or low-cost commercial graphical interfaces to these programs. For example, *SourceTree* (Atlassian, 2016b) is a free GUI client for *Git* that runs on both *Windows* and *Mac*. Several online hosting services (GitHub Inc., 2016b; Atlassian, 2016a) have arisen that make it easy to set up version control systems for collaboration with distributed team members.

Example: Consider two building energy modelers, Mary and Joe, who are working collaboratively on a large model. Figure 3 shows a screenshot of the *SourceTree* graphical interface to the *Git* version control system from Mary's computer. The view in Figure 3a is split into four sections. From top-right going clockwise we have:

- A *diff* view of Mary's new changes. Green lines preceded by "+" indicate additions to the project.
- A *commit* log entry box where Mary can explain what her changes are and why she made them.
- A view of her *working directory* on her file system. Note that the file "eem_measures.docx" has not been added to version control because it is not part of the same "logical" change that Mary is committing to the project.
- A *staging area* representing the changes that will constitute the next "snapshot" to be committed to Mary's change history.

Figure 3b depicts the history of snapshots as seen from Mary's repository. Time is depicted as advancing forward from bottom to top with each dot indicating a snapshot of the project's files at that point in time. The colors correspond to the team member who created the given snapshot. We can clearly see the times when Joe and Mary are working in parallel and we can also see the times when their changes are merged back into a single timeline. Merging is often handled automatically by the system with human intervention only required when two people change the exact same lines. By clicking on any of the snapshots in the history, Mary can get a detailed view of exactly what changed, by whom, and for what reason. Not only does this give us a means to track changes and audit prior work, we just can't imagine doing collaborative work on the same model without version control.

Templates

Templates and template systems arose out of the need for programmers to develop web pages with "dynamic content"—that is, content that updates frequently or changes for each user. The programmers realized that while this dynamic content could be different each time, the basic structure of the web page stayed the same. To separate the "things that change" from "the things that stay the same", they came up with the idea of templates.

A template system consists of a template engine, template files, and data to fill the templates with. For our purposes, templates are plain text files with "holes" in them to be filled externally by some data source. The process of filling a template's "holes" with dynamic data is called "rendering". Template engines are computer programs that render templates given some data. Templates can be written in various template languages. A template language yields the syntax to specify the "holes" in the templates and how to fill them. Templates support the "DRY" (Hunt and Thomas, 1999) principle used by computer programmers: "don't repeat yourself". DRY is a principle that prescribes that all unique information is only defined once and then referenced thereafter as opposed to redundantly repeating the information—repetition creates maintenance challenges.

Relevance: When we look at our building energy simulation input files, regardless of the tool and graphical interface, we can't help but see the massive amounts of repetition present in our input files. Templates offer us a way to reduce this repetition and separate the parts that change from the parts that stay the same.

Example: Consider some typical input data that declare three thermal zones for a model. Due to page constraints, let's pretend the input is represented with three trivial *EnergyPlus* objects, each having only one field. In reality, each object would be much larger with many fields. However, all of the fields except one has exactly the same input data—only the zone name is changing between objects.

To give you a sense of what templates can do, examine Figure 4 which shows a pair of template files using the Params template system (Ellis, 2015). The file "zone.pxt" is a text file that takes one "hole" (or "parameter"), called "name". The file "master.pxt" then inserts the zone template three times using three separate values for "name" with the result being the output of rendering the "master.pxt" file.

Scripting

"Scripting" refers to the act of writing small programs that automate the execution of tasks that could have been manually run one after the other by a human operator. The programming languages used for scripting are often dynamic—offering instant feedback—and tend to be easier to learn than languages used for general purpose programming such as Java and C++. Scripting languages tend to be well suited for tasks such as connecting the output from one application to the input for another application. They are also very effective for text processing and file system operations. Some examples of scripting languages include Python, Ruby, Bash, Windows DOS Batch, Visual Basic, and JavaScript.

Zone Template File: zone.pxt

```
<%#INITIALIZE
  parameter "name" # Name of the Zone
#>
Zone, <%= name %>; !- Name
```

Master Template File: master.pxt

```
<%=insert 'zone.pxt',:name=>'Room_Floor_01'%>
<%=insert 'zone.pxt',:name=>'Room_Floor_02'%>
<%=insert 'zone.pxt',:name=>'Room_Floor_03'%>
```

Rendered Result

```
Zone, Room_Floor_01; !- Name
Zone, Room_Floor_02; !- Name
Zone, Room_Floor_03; !- Name
```

Figure 4. Template Example Using Params

Relevance: Scripting is the key to automating repetitive tasks. Because simulation engines often use text files, there are numerous opportunities for automating the workflows for modeling on both input and output processing. Some of the possible uses for scripting in the context of modeling include:

- Coordinating the running of multiple simulation tasks
- Preprocessing inputs
- Converting between data formats
- Generating input files from templates
- Performing parametric sweeps and/or optimization
- Postprocessing of results including generating figures and tables.

Even if your simulation engine does not use a text-based input file, there may still be ways to automate the processing of files and running of applications.

Getting Started: Scripting is a large topic. It is beyond the scope of this paper to actually teach you a programming language. However, we'd like to point you in the right direction.

For the purposes of building energy modeling, we find two languages worthy of mention: Python and Ruby. The Python programming language was first released in 1991 by Dutch computer scientist, Guido Van Rossum. Python is used extensively by scientists and engineers and contains many numerical computing and plotting libraries. The *Eppy* project is a good example of using Python to script the input and output of *EnergyPlus* (Philip, 2016).

Ruby was first released in 1995 by Japanese language designer, Yukihiro "Matz" Matsumoto. Notably, Ruby is the scripting language used by both *OpenStudio* and *SketchUp*. In particular, Ruby can be used to create "measures" in the *OpenStudio* environment which automate certain pre-processing, post-processing, and model- development tasks (NREL, 2016).

Numerous books on Python and Ruby have been written and are available from various online book sellers. In addition, many free online courses and other resources exist (see, for example, Codecademy, 2016a, 2016b).

ADVICE ON LEARNING

Some of the challenges for modelers interested in adopting the tools and techniques described here can include lack of time and resources, organizational inertia, fear of the unknown, and information overload. We can't help with the first three, but we can help you combat information overload:

- Try one of the topics on an upcoming project
- Only attempt one topic at a time
- Don't try to learn everything at once
- Learn just enough to be productive
- Don't stay stuck; use available resources (see below)
- Don't worry about making mistakes
- Try something!

If you need help, your best resources are friends and colleagues who know the tools already, your company's IT department, training courses, books and websites, and online question-and-answer sites such as Stack Overflow (Stack Exchange Inc, 2016) and Unmet Hours (Big Ladder Software, 2016).

CONCLUSIONS

This paper presents four categories of tools and techniques from the software development field that we

believe can be successfully adopted by building energy modelers to improve productivity. All of the topic areas are based on principles that originate as far back as the 1970s. These are ideas that have been forged through the tests of time and proven to be useful. The tools and techniques we present have a high adoption rate among professional computer programmers today. We have made the argument that modelers are suffering from a lack of knowledge of these topics and we hope to change that.

Although the methods here work best with modeling tools based on text-based input files, you can still apply the advice on version control and scripting even if using a non-text-based program.

We began this paper with an admonition that the accidental complexity in our tools and workflows is hurting our productivity and stealing away time for critical thinking. We believe the approaches presented here, when applied thoughtfully and strategically, will help you minimize that accidental complexity and become a better modeler. Although learning something new requires an upfront investment in time, you will reap the benefits of your effort many times over with greater productivity. The most significant gains will be realized when an entire team can adopt new tools and workflows that promote collaboration. While the topics presented here offer some good first steps, they only scratch the surface of how to "model like a programmer".

We hope this paper will serve as a starting point for enhancing your own work, and perhaps spark a discussion on these topics at your organization.

REFERENCES

Atlassian. 2016a. "BitBucket Collaborative Hosting Service." https://bitbucket.org.

———. 2016b. "SourceTree Graphical Client Website." Atlassian. https://www.sourcetreeapp. com/.

Big Ladder Software. 2016. "Unmet Hours: Question-and-Answer Resource for the Building Energy Modeling Community." https://unmethours. com/questions/.

Brooks, Frederick P., Jr. 1987. "No Silver Bullet— Essence and Accidents of Software Engineering." Computer 20 (April). IEEE: 10–19. http://faculty.salisbury.edu/ %7exswang/Research/Papers/SERelated/no- silver-bullet.pdf.

Chacon, Scott, and Ben Straub. 2014. Pro Git. 2nd ed. Apress. https://git-scm.com/book/en/v2.

Codecademy. 2016a. "Python." https:// www.codecademy.com/learn/python.

———. 2016b. "Ruby." https://www.codecademy. com/learn/ruby.

Ellis, Peter. 2015. "Parametric Modeling with Templates and Scripting." Presentation at 2015 ASHRAE Energy Modeling Conference, Sep. 30 - Oct. 2, 2015, Atlanta, GA. http:// bigladdersoftware.com/projects/params/.

GitHub Inc. 2016a. "Atom Text Editor Website." https://atom.io/.

———. 2016b. "GitHub Collaborative Hosting Service." https://github.com.

Helios Software Solutions. 2016. "TextPad Text Editor Website." https://www.textpad.com.

Ho, Don, and Notepad++ Collaborators. 2016. "Notepad++ Text Editor Website." https:// notepad-plus-plus.org/.

Hunt, Andrew, and David Thomas. 1999. *The Pragmatic Programmer: From Journeyman to Master*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

NREL. 2016. "About Measures." National Renewable Energy Laboratory. https://nrel.github.io/ OpenStudio-user-documentation/getting_ started/about_measures/.

Philip, Santosh. 2016. "Eppy Project." https:// github.com/santoshphilip/eppy.

Scooter Software Inc. 2016. "BeyondCompare Website." http://www.scootersoftware.com/.

Sink, Eric. 2011. Version Control by Example. On- line Book. http://ericsink.com/vcbe/index. html.

SourceGear. 2016. "DiffMerge Website." https:// sourcegear.com/diffmerge/downloads.php.

Stack Exchange Inc. 2016. "Stack Overflow: On- line Question and Answer Site." http:// stackoverflow.com/.

The Software Freedom Conservancy. 2016. "Git Version Control System Website." The Software Freedom Conservancy. https://git-scm.com/ .

Tupper, Kendra. 2011. "Building Energy Modeling Innovation Summit: Post-Report." Rocky Mountain Institute. http://www.rmi.org/ Content/Files/BEM_Report_FINAL.pdf.

WinMerge Developer Team. 2013. "WinMerge Website." http://winmerge.org/.