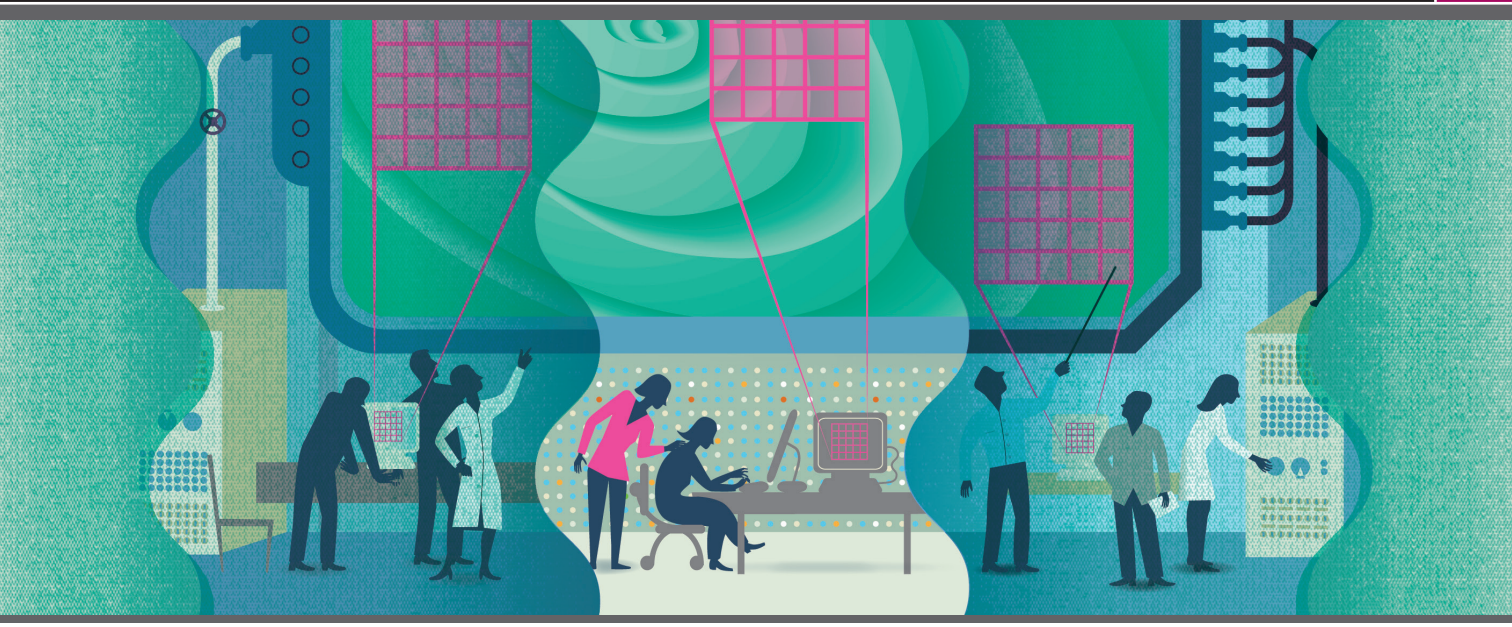

Adapting Collaborative Software Development Techniques to Structural Engineering

Peter Sempolinski, Douglas Thain, Zhigang (Daniel) Wei, and Ahsan Kareem | University of Notre Dame

A prototype virtual wind tunnel implements some of the ideas of collaborative software design in the context of structural engineering, letting users edit structural geometries, compile geometries into usable models for simulation, perform wind simulations, evaluate the results, and then share and discuss their work with other users.

In recent years, software development has benefited greatly from the tools used in collaborative workflows that make it easier to share, compile, test, analyze, and track changes to source code. These tools simplify coordination among developers, even across huge distances, which in turn lets users track problems, record code history, give contributor credit, and facilitate discussion among software developers. Such actions aren't just beneficial to software developers but to other kinds of designers as well.

Software developers around the world can have communal access to the same software code base in a repository. They can contribute independently created solutions to a project, track issues and contributions, and test their products in iterative cycles of continual testing and revision. All of this serves to create a virtual community of software developers mediated by software technology. Such tools move to a whole new level of potential when combined with the Internet's ubiquitous power, which greatly expands the scope of the community contributing to any particular project.

Related Work in Software-Mediated Collaboration

The success of collaborative techniques can, and already is, being extended in domains other than software. The system opencores.org, for example, helps users design microprocessor architectures in an open source way, extending ideas from software to hardware. In the scientific community, the myExperiment framework lets users "find, use and share scientific workflows and other Research Objects, and to build communities."[1] It's a means of sharing, storing, and discussing experimental workflows and other projects.

Our work has been connected with structural engineering, for which there's also interest. OpenSEES (http://opensees.berkeley.edu) is an open source tool for exploring the simulation of earthquake resistance. When it comes to collaboration, the Google Sketchup 3D Warehouse (https://3dwarehouse.sketchup.com), an online library of designs, lets designers upload and share design files. We focus on wind simulation, but others are considering how to expand the finite analysis of structural components to cloud computing resources.[2] Within scientific computing, a great deal of effort has gone into making complex software tools available online. Prominent among this work is HUBzero,[3] the back end for NanoHub, a platform for nanotechnology simulations. These are just a few examples of potential avenues for collaborative, software-mediated design to be embraced outside of software engineering.

References

1. D. De Roure, C. Goble, and R. Stevens, "The Design and Realisation of the Virtual Research Environment for Social Sharing of Workflows," *Future Generation Computer Systems*, vol. 25, no. 5, 2009, pp. 561–567.
2. J. Juzna et al., "Solving Solid and Fluid Mechanics Problems in the Cloud with Mosaic," *Computing in Science & Eng.*, vol. 16, no. 3, 2014, pp. 68–77.
3. M. McLennan and R. Kennell, "HUBzero: A Platform for Dissemination and Collaboration in Computational Science and Engineering," *Computing in Science & Eng.*, vol. 12, no. 2, 2010, pp. 48–52.

But the potential for forming collaborative endeavors mediated by software tools isn't just limited to software design. In structural engineering, for example, people endeavor to design structures of various types, many of which are constrained by size, aesthetics, floor plan, wind loading, and legal compliance, and so on. Although software tools are available for engineers to elaborate designs, simulate them under adverse conditions, or visualize their appearance, few of these software tools are integrated to work with each other. Moreover, the various relevant ideas, such as design, architecture, fluid mechanics, or law, constitute different and often disjoint knowledge domains. In addition, there's a cultural gap: structural engineers aren't as accustomed to software-mediated collaboration as software developers are. And many of the complex analytical tools are computationally intensive, requiring familiarity with and access to high-performance computing resources.

In this work, we demonstrate how to confront these challenges to apply the techniques of collaborative software design to the design problems found in structural engineering. As part of that demonstration, we developed a virtual wind tunnel (VWT), a prototype that simulates wind effects on structures in a collaborative setting. This work involved the challenging task of combining a seamless user experience and ideas from domains of structural design, high-throughput computing, geometry, complex simulation, Web interfaces, and more. In particular, moving structural design ideas from the high-level domain of user input to the low-level domain of simulation required a complex act of compilation. Our work, however, is only a small portion of the developing ecosystem of software tools now being created to expand the scope of collaborative design. We advocate further developing this ecosystem so that structural design can benefit from the same wide-scale software-mediated collaboration that has become ubiquitous in software design.

The Collaborative Design Loop

Collaborative design, while used in many settings, is most closely associated with large open source software projects developed by hundreds or thousands of contributors in a worldwide virtual community. When we speak of "collaborative design," we describe how an ecosystem of tools allows for software design projects in a wide variety of settings and developer communities. Some of the common features of this kind of collaboration include the following:

- *Source control*. Instead of having every person involved in a software project operate on unorganized, individual copies of code, source control programs such as Git or SVN allow changes to code to happen in a central location. This allows changes to be tracked, credited, commented upon, and, if needed, reverted. It also makes it easier to combine the work of several people contributing simultaneously, especially if they aren't all working on the exact same portion of code. This reduces problems such as one programmer's work conflicting with or inadvertently destroying another programmer's work.
- *Automated evaluation*. In many software projects, it's useful to quickly build and test software to immediately determine the quality and correctness of any changes. Automated programs

can compile the source and run it against a set of test cases at specified intervals, providing an independent and objective means of verifying the correctness of any changes to code.

- *Iterative workflow*. In recent years, rather than reaching a final "done" state, the mark of successful software is its continual revision. Software is tested, and new features are suggested as bugs are found, which prompts further changes that motivate new versions of the software. Users like the software enough, in spite of its bugs, to send bug reports. Developers maintain interest and contact with each other. This continual revision is facilitated by many kinds of tools, such as email and bug-tracking databases.
- *Shared results*. In collaborative software design, virtual communities (which are also mediated by software) can arise. These communities discuss the results of what they've created, from something as simple as, "It didn't work," to careful evaluations of specific code revisions. This discussion can take place in ways directly tied to the software code and the programs that store and manage it—for example, a bug report can be directed to the developer who wrote a certain piece of code because the version control system has a record of that contribution.

No single tool accomplishes all of the above—rather, there's an entire ecosystem of tools. Efficient compilers allow for repeated recompiling of code and automated evaluation. Programmers can contribute changes to a software project using version control programs. Community tools and repositories, such as SourceForge, can provide additional features. One such tool, GitHub, hosts thousands of projects, large and small.[1] Of course, tools as simple as email and electronic documentation (such as custom wiki pages) also have roles to play in helping software contributors work together.

All of this makes open source and collaborative techniques highly flexible. At one extreme, two coders working on secret code for a small startup communicate by email, using SVN to record changes and avoid breaking each other's work. At the other extreme, a large, worldwide open source project is hosted on GitHub and incorporates all the features of that platform. In both cases, the software developers use tools to make it easier to contribute work to a common project. Many of the most commonly used software programs embrace collaborative design, at least to some extent. To give two examples, the Linux kernel was, in fact, the impetus for Git's development, and the Apache webserver has, for many years, been the most commonly used program for hosting webpages on the Internet and prides itself on its worldwide, collaborative development.

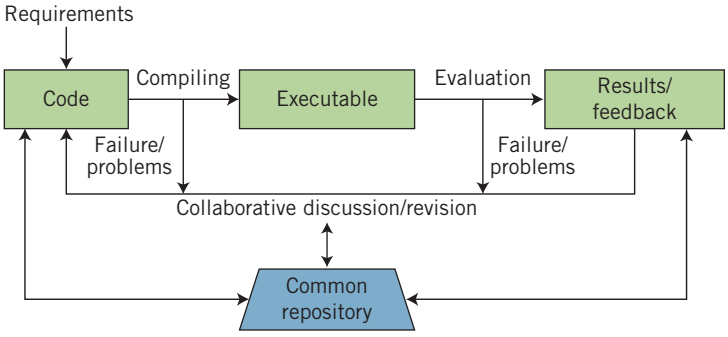

Figure 1. The key aspects of communal design for software. Arrows represent compilation, evaluation, and collaboration, and source code is passed from coders to the compiler, executables from the compiler to testers, and user comments and test results back to the coders.

As Figure 1 shows, software design can be thought of as three key stages in an iterative design process. Arrows represent compilation, evaluation, and collaboration, and source code is passed from coders to the compiler, executables from the compiler to testers, and user comments and test results back to the coders. Ideally, the code, executables, and comments are all stored in a common repository with its own software to facilitate the storage, backup, and retrieval of contributions and their history.

We propose an analogy for these key stages in the workflows of structural engineering and other domains. Figure 2 shows this analogous workflow, with each stage having its own facilitating software. In the software domain, software code is the primary means of modifying the end product; in structural design, the end product is a building design, so files contain information about the structure's geometry. To perform coding, or design, on such structures, we use CAD tools, such as AutoCAD and Sketchup, and to test a structural design, we perform simulations. For commenting, we need a way for developers to view the buildings and simulation results, and then communicate with each other about them.

If simulation must take the place of software testing, then we must also have some way to bridge the gap between the building description and the simulator. To bring iterative, collaborative design into the engineering domain, an equivalent compiler must be available. In software, the input to a compiler is source code, and the output is a runnable program. In structural engineering, the input needs to be a description of a building geometry,

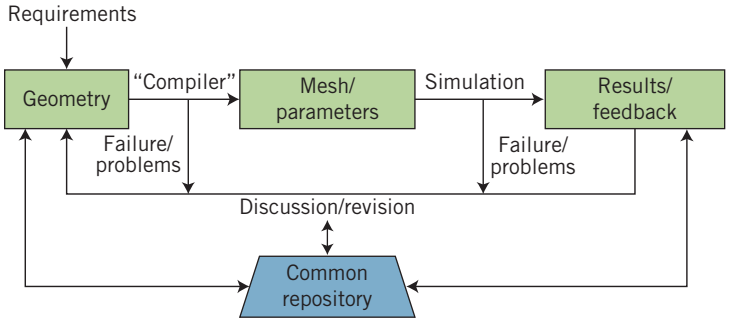

Figure 2. Collaborative design within the structural engineering domain. The end product is a building design, so files contain information on the structure's geometry. To perform coding, or design, on such structures, designers use CAD tools, such as AutoCAD and Sketchup. To test a structural design, they perform simulations.

and the output is the file suitable for some simulator. We can imagine an idealized system in which an architect could compile his or her CAD design into the files necessary for earthquake simulators, legal building code analysis, wind simulators, and the like. But because each of these simulators is different, each one needs its own compiler—and this is only one of many translating programs needed to make a truly collaborative structural engineering workflow function. Also needed is a common repository, with ways of storing each stage of the process for analyzing and sharing.

Let's examine how the features of collaborative software design that we discussed earlier can transition to the engineering design domain:

- *Source control*. The "source" now takes the form of a building design. In spite of this, source control can resemble the software paradigm—for example, a system that records changes and contributions can store specific building designs.
- *Automated evaluation*. The building source must be simulated for a wide variety of conditions, so automated evaluation is complicated, but it's also essential. We can't expect architects to be familiar with the nuances of complex simulation software, so multiple aspects of the simulation must occur somewhat automatically, giving building designers a more fluid experience in submitting a design and getting immediate insight.
- *Iterative workflow*. The focus of the iteration is now different. Rather than build-test-rebuild iterations, as in software, we must perform design-simulate-redesign. But the idea is the same—the proposed design is still tested and revised in a cycle of re-evaluation.
- *Shared results*. The challenge of sharing results is now focused on giving designers the ability to visualize simulation results, which requires tools that let engineers see pictures, graphs, and other images that correspond to visuals of their designs and simulation results.

Of course, buildings aren't software, so the nature of the final product in software design, in contrast to structural engineering, is very different. Because software exists primarily in computer memory or hard drives, it's relatively easy to replace one version of a software product with a newer, better one. This makes the iterative design process relatively intuitive, as updating a piece of software doesn't usually involve anything more that transmitting and installing the new version. Compare this with the final product of structural engineering—a building, which is very expensive to construct and can't simply be torn down and replaced with a new version without considerable effort and expense. Also, a constructed building can't be copied or retransmitted like software. However, the extreme expense of building and rebuilding in structural construction is all the more reason to make it easier to simulate the results of construction, as well as to make it easier for more collaborative workers to coordinate with each other to evaluate designs.

Ultimately, our goal isn't software tools, but to use software tools for better and more abundant engineering. Similarly, software design doesn't use collaborative software for its own sake, but to enable better software in a variety of settings, including open source settings or design by talent made available throughout the world. We propose that structural engineering can, and already does, benefit in similar ways, with collaborative design expanding the different ways that structural engineering can be done, such as in open source contexts or other new ways.

Case Study: The Virtual Wind Tunnel

We've been exploring the potential of these ideas in the context of the OSDCI (or Open-Sourcing the Design of Civil Infrastructure) project (see the sidebar, "The OSDCI Project"). This project explores many aspects of bringing the techniques and tools of structural design into a collaborative and open source context. As part of this project, we built a VWT as a means of exploring the intersection of collaborative design, complex simulation, civil infrastructure, and computer science. In addition, our VWT flexibly incorporates the use of high-performance computing and mechanisms for

data reproducibility. We've already used this system for various case studies and educational activities.

Our VWT is tuned to explore the simulation of wind against tall buildings. From the user's perspective, it's comprised of a Web portal that lets users upload building shapes and perform wind simulations on them. These wind simulations are performed with industry-grade computational fluid dynamics (CFD) software, specifically OpenFOAM (www.openfoam.com), using a computing back end that deploys various clouds, grids, or clusters in a high-performance computing setting.[2] Within this portal, we made it possible to comment on and critique other collaborators' work. Figure 3 shows screenshots of the system in action, corresponding to the various stages in an iterative, collaborative design. Because structural design has many different considerations, an idealized ecosystem of design tools, analogous to software design, would have to include not just our work on wind effects but tools for considering aesthetics, material cost, earthquake resistance, floor plan design, legal compliance, and more.

The VWT has three parts: a computing back end, a collaboration front end, and a CFD compiler. The front end is a Web-based application, primarily written in PHP, that's responsible for interacting with users. It authenticates users, receives uploaded building shapes, records specified simulation parameters, passes simulations to the back end, displays simulation results, and provides a forum for user interaction. This information is passed to the rest of the system via a database and parameter files.

The back end receives the simulations to be run and passes them to actual computing engines, such as grids or clouds. We use a framework called Work Queue to assist in dispatching tasks across clusters, clouds, and grids.[3] We use a combination of an on-campus grid-computing engine and a set of virtual machines in a private cloud. We've also experimented with dispatching simulations to other experimental computing infrastructures, such as FutureGrid. In this way, we make available powerful computing resources to people whose domain of knowledge lies outside computer science. The back end is also responsible for passing the various dependency executables (such as the OpenFOAM simulator itself) to the site of computation. This lets us use a wide variety of computing resources to support the VWT system.

There were multiple challenges in building the VWT back-end systems, one of which was how to manage the deployment of software dependencies because the CFD software was deployed to high-performance computing engines not under our control. Moreover, such software isn't installed on many clusters and requires a particular filename space and environment to run. To solve this problem with minimal overhead, we developed a system of software suitcases that contain the dependency files for various CFD tasks. Upon starting a task on a machine, a lightweight porter program unpacks the suitcase into a known folder. The porter then manipulates its own environment variables and starts the actual CFD task as a subprocess. The manipulation of the file space and environment directs the CFD task to its critical files, which could include shared libraries, Python modules, or other program executables.

The Missing Link: A Compiler

The CFD compiler, however, was the most challenging component. Throughout this work, we found that collaboration among software tools was a key facilitator of collaboration among people. In this case, to bridge the divide between the domains of

The OSDCI Project

This work was conducted in the context of a wider project on Open-Sourcing the Design of Civil Infrastructure (OSDCI), funded by the US National Science Foundation. It let us use parts of the virtual wind tunnel (VWT) as it developed as the basis for other experiments in crowdsourcing structural engineering tasks. For these experiments, we explored whether individuals in the general public (the "crowd") could, with minimal training, recognize simulations as either being potentially correct or incorrectly performed.[1]

The larger OSDCI project is concerned with finding ways to address problems of structural design in open source settings. Part of that endeavor requires producing collaborative frameworks for open source design—other projects include testing if crowdsourced contributors can identify various types of structural damage from photographs of damaged buildings. One of the envisioned broader impacts of OSDCI is to facilitate faster reconstruction after a natural disaster, such as an earthquake, by widely encouraging open source and collaborative work in the problems of reconstruction.

Looking forward, our long-term vision is for building designers to have available a whole suite of software tools, each with its own facility for user-to-user interaction, to build, analyze, or critique multiple aspects of designs, collaboratively. To do this, however, further collaboration among people must be realized—by furthering the collaboration among new software tools, those that already exist, and those yet to be built.

Reference

1. M. Staffelbach et al., "Lessons Learned from an Experiment in Crowdsourcing Complex Citizen Engineering Tasks with Amazon Mechanical Turk," *Collective Intelligence*, 2014; http://arxiv.org/abs/1406.7588.

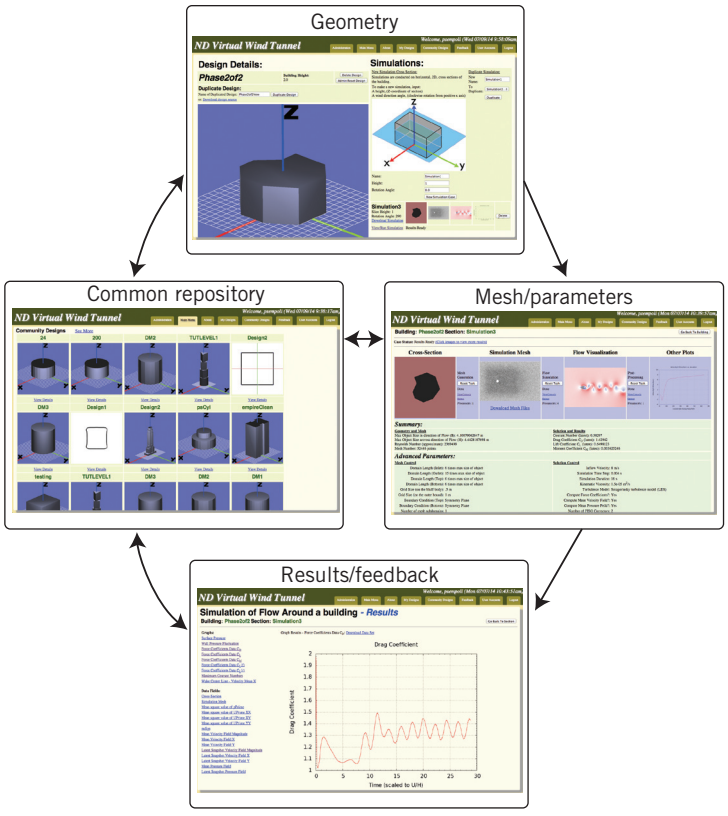

Figure 3. Prototype virtual wind tunnel. This VWT has three parts: a computing back end, a collaboration front end, and a computational fluid dynamics compiler.

structural design and fluid mechanics, which are two very different fields, we had to bridge the distance between design software and fluid simulation software. In particular, CFD simulation is very difficult for newcomers to fully understand. In addition to the raw data about the object being simulated, CFD requires detailed information about the arrangement of the simulation domain, mostly represented as the simulation's mesh, a discretization of the simulation domain—each cell in the mesh is a single point where the pressure and velocity will be computed at each time step. In effect, the simulator requires a low-level description of how the simulation is to run. The process of compiling, like in software code, wasn't merely an act of translating equivalent expressions of information from one form into another. Rather, we had to capture the essential structural information of the high-level inputs into an intermediate form and then incorporate new details pertaining to the low-level expression of the simulation.

Our front end was able to receive COLLADA or .dae files as a building shape (this fairly widespread format can be extracted from Google Sketchup and other programs), along with the needed simulation parameters. From this high-level description of the building, we had to compile a low-level description of the simulation in the form of a mesh and the OpenFOAM input files. This process involved several programs, including an unstructured mesher called Gmsh.[4] The inputs to the front end were translated into distilled files containing the essential geometric information that could then be re-expressed as input for gmsh, which created an unstructured mesh grid around the object. (As of right now, our simulator works on horizontal cross sections of the building.) This mesh was translated into OpenFOAM's mesh format, with additional files for other simulation parameters. In effect, by translating from one set of tools to another, we found ways to open up CFD tools to non-CFD-experts in specific situations, in spite of the high degree of expertise normally needed just to run CFD simulations.

Figure 4 shows how the raw input is a high-level description that's compiled with more information to become a lower-level description appropriate for the simulator. Because programmers don't have to understand the workings of a software compiler or the hardware to write code, our system's users don't have to understand our compiler's inner workings or the simulator that runs the simulations. This prototype is an example of how software can provide a transition from building design to building testing.

Together, the software components in our prototype form a tool that lets users upload a building shape and select cross sections for simulation with various parameters. Other users can then comment on those simulations and the design, leading to further revisions until everyone's satisfied with the results. If this idea were extended to its fullest realization, we would need compilers not just for wind but for other design considerations as well, some of which we've already mentioned. A building must be evaluated for many considerations, which requires compiling for many different types of simulations and visualizations. This is similar to a piece of software designed to run on multiple platforms, thus it must be compiled in many ways.

Record, Reproduce, and Reanalyze

One of the key benefits of collaborative software design is the relative ease by which contributions and results can be reproduced, catalogued, and traced. In the software design domain, this is accomplished as part of the version control system. Commits to the code repository typically include records of the changes and the person responsible, which makes it relatively easy to reproduce

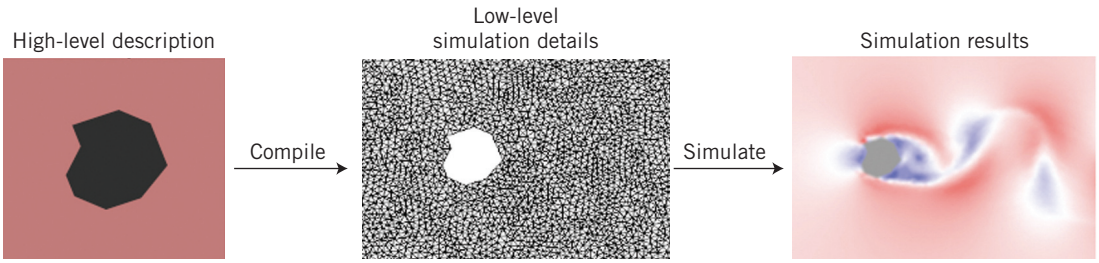

Figure 4. The progression from high-level description to detailed simulation input to results. Because programmers don't have to understand the workings of a software compiler or the hardware to write code, our system's users don't have to understand our compiler's inner workings or the simulator that runs the simulations.

a software product in a particular state or to trace contributions.

In our own work, however, we found more avenues for results to branch from other results. For example, a particular building might be simulated at many different cross sections, and one of those cross sections might involve different mesh configurations. A particular mesh configuration might have different simulation parameters, including duration, time steps, and wind velocity, so we had to capture the many paths against which we could preserve and reproduce results.

To allow such result preservation, users can copy simulations from other users and modify them. A record is preserved of the link to the previous simulation, allowing back-tracing through the history of a particular set of simulations, with users' names attached. When copying a simulation, a user can choose to rollback the copy to any stage in the computation and make changes from there, allowing new work to branch off from any point of departure. Any result is stored alongside the simulation parameters and building design that produced it, allowing for clean reproducibility of results. Raw simulation files and results can be downloaded, allowing further independent reanalysis of any data.

These mechanisms allow for significant flexibility when it comes to managing data, results, and contributions. Figure 5 shows how a second user can re-examine a particular result. User A proposes a building cross section and simulates it using a particular mesh. User B duplicates this workflow twice, with the system noting the relationship to the original. User B then tries new simulations—in this case, one with a new mesh and another with a different wind velocity.

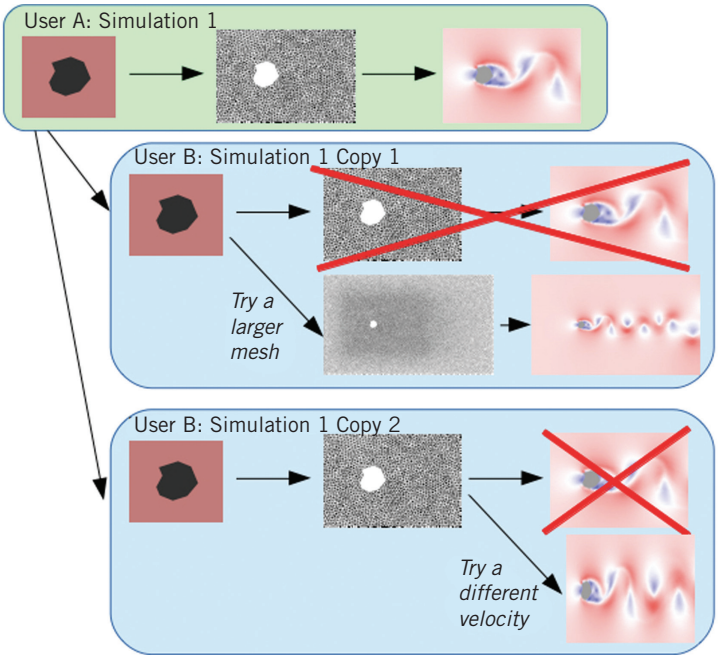

Figure 5. A user proposes a new take on a previous simulation. User A proposes a building cross section and simulates it using a particular mesh. User B duplicates this workflow twice, with the system noting the relationship to the original. User B then tries new simulations—in this case, one with a new mesh and another with a different wind velocity.

Through this work, and in exploring many other similar endeavors, we've learned about the potential of, the challenges in, and the need for extending collaborative techniques to the structural engineering field. So far, we've used the VWT for various educational and research projects, including a multipart crowdsourcing study conducted with the VWT as a platform for generating simulation results, both by the experts conducting the study and its participants. Among meshing, simulation, and postprocessing, these participants ran more than 3,000 tasks on the VWT. Although we're only just beginning to explore this problem space, our research has attracted engineering experts, college

students, and untrained "crowd" users, all of whom accessed and used the VWT, leading us to wonder what other tasks in structural design could be opened to further collaboration and to what degree.

We also had the opportunity to use our system in a classroom setting to give graduate students hands-on experience with CFD simulations. A CFD expert used this opportunity to write a short tutorial explaining some of the finer aspects of CFD analysis, which the students had an opportunity to experience and experiment with using an actual CFD simulation. Their feedback played an essential role in revising and expanding our VWT system.

There are, however, challenges remaining. Using software to design software is natural—programmers are comfortable with software in general—but we can expect engineers to approach their work with a different mindset. Introducing software collaboration to structural engineers requires understanding how their mindsets can be served, rather than opposed, by software. The next logical step in developing this idea would be to pull together many different existing simulation tools into a single system and explore how to convey useful information without overwhelming users.

We're interested in marshaling all types of competent knowledge in the pursuit of better structural design, which has a direct and wide-ranging human impact. For example, large-scale natural disasters of every kind occur throughout the world practically every day. To rebuild infrastructure and communities in the aftermath of such an event it would be beneficial to allow qualified engineers throughout the world to contribute ideas, designs, and constructive criticism. Similarly, as global environmental concerns become more critical, it's expected that architects and environmental scientists will need to work more closely together. We want to help bridge the collaborative gap between these domains. What we've described here, both of our own work and other work also being done along these lines, is the first step in an effort that we hope and expect will yield greater impact as more avenues for collaborative design emerge. ■

Acknowledgments

This work was supported in part by the National Science Foundation under grants OCI-1148330, CBET-0941565, CNS-0855047, and CNS-0643229.

References

1. L. Dabbish et al., "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository," *Proc. ACM Conf. Computer Supported Cooperative Work*, 2012, pp. 1277–1286.
2. P. Sempolinski et al., "A System for Management of Computational Fluid Dynamics Simulations for Civil Engineering," *Proc. 8th IEEE Int'l Conf. eScience*, 2012; 10.1109/eScience.2012.6404433.
3. P. Bui et al., "Work Queue + Python: A Framework For Scalable Scientific Ensemble Applications," *Proc. ACM/IEEE Int'l Conf. High Performance Computing, Networking, Storage, and Analysis*, 2011.
4. C. Geuzaine and J.-F. Remacle, "Gmsh: A Three-Dimensional Finite Element Mesh Generator with Built-In Pre- and Post-Processing Facilities," *Int'l J. Numerical Methods in Eng.*, vol. 79, no. 11, 2009, pp. 1309–1331.

Peter Sempolinski is a graduate student in the Department of Computer Science and Engineering at the University of Notre Dame. His research interests include human-computer interaction, system design, and security design. Sempolinski received an MS in computer science from Notre Dame. Contact him at psempoli@nd.edu.

Douglas Thain is an associate professor in the Department of Computer Science and Engineering at the University of Notre Dame. His research interests include open source software, clusters, grids, clouds, and high-performance computing. Thain received a PhD in computer sciences from the University of Wisconsin–Madison. Contact him at dthain@nd.edu.

Zhigang (Daniel) Wei is a computational fluid dynamics researcher at Boeing (China). His research interests include CFD techniques, high-performance computing, and software development in these areas. Wei received a DPhil in bridge engineering from Tongji University. He worked as a postdoctoral scholar in the Natural Hazard Modeling Lab in the Department of Civil and Environmental Engineering and Earth Sciences at the University of Notre Dame during the time of this work. Contact him at zwei1@nd.edu.

Ahsan Kareem is the Robert M. Moran Professor of Engineering at the University of Notre Dame. Kareem received a PhD in structural engineering from Colorado State University. He has been awarded numerous honors, including the Presidential Young Investigator Award from the White House Office of Science and Technology, the American Society of Civil Engineers' von Karman, J. Croes, J.E. Cermak, and R.H. Scanlan Medals, and the International Association for Wind Engineering's A.G. Davenport Medal for contributions to dynamic wind load effects on structures. Kareem is a member of the US National Academy of Engineering, a Distinguished Member of ASCE, and a member of IEEE. Contact him at kareem@nd.edu.