# The Tool(s) Versus The Toolkit

Chris Mackey[1,2(✉)] and Mostapha Sadeghipour Roudsari[1,3]

[1] Ladybug Tools, Boston, USA
{chris,mostapha}@ladybug.tools
[2] Payette Associates, Boston, USA
[3] University of Pennsylvania, Philadelphia, USA

As the architectural practice continues into the digital age, most designers agree that new computational technologies should be harnessed for the betterment of their buildings and workflows. However, there are currently several competing philosophies for how such technologies should be best integrated into the design process. Until recently, much of today's practice has found itself leaning toward one of two camps: one that allows for a distributed yet disconnected approach and another that strives toward a centralized methodology. While each philosophy has its strengths, they both suffer from significant limitations.

Perhaps one of the clearest examples of the distributed and disconnected approach is the current state of environmental performance software, which has seen the introduction of countless new specialized tools over the last few decades. The goals of these distinct applications range across the board from daylight/glare modeling, to HVAC sizing, to full-building energy simulation, to thermal comfort forecasting, to embodied carbon estimation, to structural member sizing, to envelope insulation evaluation, to condensation risk mitigation, to acoustic modeling, to stormwater/rain collection management (Moe 2013). The list goes on and, in order to engage the full range of topics necessary for good environmental design, practitioners frequently find themselves recreating models in each of these different interfaces. As a result, designers are typically unable to account for all of these criteria in the scope of their projects and this additive approach of piling more tools onto the process becomes over-complicated and inefficient.

Many have recognized this trend in recent years and have attempted to respond to the issue. Perhaps the clearest example of this is the rise of building information modeling (BIM), which anticipates a streamlined design process by having all design team members put their data into a single model built with one software package (Negendahl 2015). While such BIM models can be useful for organizing and documenting final designs, their sheer size can make them inflexible and difficult to iterate upon. The more that is added to a model, the harder it becomes to change or test out new ideas with it. If one were to support all of the previously listed environmental analyzes with a single central model, each model element would have a huge number of properties and geometry types associated with it. This is because the data that is needed for one type of study, such as energy modeling, is often very different than that needed for another study, such as structural member sizing. So adding/changing any given element of such a central model would either be time consuming and unfriendly


K. De Rycke et al., *Humanizing Digital Reality*,
https://doi.org/10.1007/978-981-10-6611-5_9

to iteration or would result in messy, poor data for certain types of studies as practitioners add building elements to satisfy only one objective at a time.

Accordingly, neither the disjointed set of tools nor the "one tool to rule them all" is a suitable solution to our dilemma of software integration. As with many dualistic situations that contrast two extremes, the best route is often a middle one that harmoniously balances the two. So what would such a harmonious balance look like for our problem of software integration? One might notice that both the disconnected and the centralized approaches suffer from the same philosophical fallacy—they focus on the tools themselves as the solution rather than the workflows or interconnection between tools. Instead of a single all-powerful tool or a disjointed set of tools to address our contemporary dilemma, a cohesive toolkit that seeks enhanced workflows between software might be far more effective. Unlike the centralized method, a toolkit would have the flexibility to engage different objectives whenever they become relevant. In other words, one does not have to specify all properties/formats of a given building element at once but can wait until one is ready to use the pertinent tool in the kit. Yet, unlike the disconnected approach, the reference to tools as part of a "kit" means that they are expected to work together in a continuous process, allowing the work done with one tool to be cleaned, formatted, and passed onto another with minimal loss of relevant data (Fig. 1).

This notion of a "toolkit" is arguably what has made visual programming languages (VPLs), such as Grasshopper (McNeel and Associates 2017) and Dynamo (Autodesk 2017), so successful in addressing software integration issues in contemporary design practice (Negendahl 2015). The fact that these VPLs break down the functions of software into discrete "components" or "nodes," makes them the literal embodiment of toolkits. Each component within a VPL has its own inputs and outputs, essentially acting as an individual tool within a larger script or workflow. As a result, users can customize their workflows based on the arrangement of components in their scripts, enabling them to engage different issues as they become relevant and experiment with new creative workflows as unique situations arise on projects (Tedeschi 2014). Furthermore, the ability to output data at different points along a VPL workflow allows plugins intended for different purposes to easily pass relevant information between one another. While VPLs are one of the more obvious examples of toolkits in contemporary

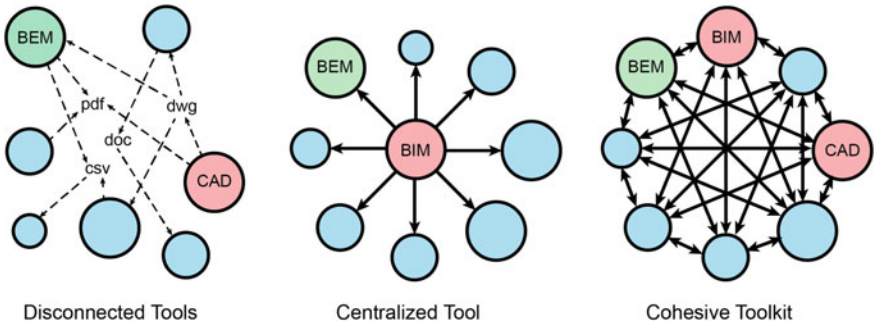


**Fig. 1** Diagrams of software integration methods

practice, the general sentiment that all software should work together can also explain why some computer applications have integrated more successfully than others with contemporary practice. Accordingly, with the goal of a toolkit in mind, the rest of this article will define key guiding principles and features that make software a part of a toolkit and therefore a particularly useful element of contemporary design processes. These "principles of the toolkit" are derived from the author's' experience developing the "Ladybug Tools" plugins (Sadeghipour Roudsari and Mackey 2017) for the aforementioned VPL interfaces. While the authors attribute much of the success of this project to these principles, it should be noted that not all must be fulfilled for a given software to act as part of the toolkit and there can be multiple pathways to addressing each principle. Ultimately, it is hoped that this list will assist both practitioners who are seeking to identify software that can be used in their toolkits as well as software developers looking to make their projects behave with this "toolkit" functionality.

## "Do One Thing and Do It Well"

Perhaps the most important feature of any software seeking to be a part of a kit is that it performs one task (or a few related tasks) exceptionally well. Many of us know from experience that our most valuable and continually-used tools are often simple in presentation. For example, many attribute the early success of Google to its founding engineers' focus on making a fast and well-indexed search engine rather than adding extraneous news, weather, and advertising images (Williamson 2005). The same can be said of many plugins in software ecosystems like those surrounding Grasshopper and Dynamo. When a large number of plugins exist within the same community, they are often forced to focus on a particular task in order to define a niche for themselves within their ecosystem. The more developers that there are in a given software environment, the stronger the need to differentiate oneself and the more intense the speciation. Even Ladybug Tools, which many people see as an umbrella for several different types of studies, has a clear boundary that defines what it does well. Specifically, this is "analysis related to climate/weather data" and, while there are many other tools related to good environmental design (like optimization algorithms, building structural solvers, and tools for creating generic charts), Ladybug Tools does not include these. Instead, if you need this functionality, we recommend that you use other tools in your "kit" that are better suited toward these tasks, like the Octopus optimization plugin (Vierlinger 2017), Kangaroo form-finding plugin (Piker 2017), or just export your data to Excel to make some generic charts. This focus on one particular task is essentially a software developer's recognition of the tool's place within a larger toolkit. As a result, a tool that is intended to be a part of a kit follows the first tenant of the Unix philosophy (Weber 2015), having fewer extraneous features and instead focusing efforts on its primary stated purpose.

## Build Interoperability with Other Tools

While it is important for software in a toolkit to focus on performing one task well, the suggestion that one "use another tool in your kit" is of little help if one cannot export one's work to such other tools. For this reason, it is critical that any software seeking to participate in a toolkit possess BOTH import and export capabilities to a variety of other formats. This is particularly relevant given that many software companies prefer to focus on importing data from a wide range of formats and neglect the development of export functionality. This is understandably the result of traditional competitive economics as companies feel that it is better to keep users within their own interfaces rather than letting them export and roam to competitor software. Yet, the adherence to this thinking often ends up hurting such software projects more than it helps since the time that could have been spent building export capabilities is instead devoted to adding features that mimic competitor functionality. Because such mimicked functionality is usually never as good as another piece of software that is dedicated to the task, there is a missed opportunity to add the most value to their work. This competitive mindset also makes it important to highlight that good interoperability not only includes the export to generic file types, like PDFs for drawings or gbXML for energy modeling, but also allows direct export to more specific formats when possible, like Illustrator for drawings and OpenStudio (NREL 2017) for energy modeling. Such direct exporting affords the smallest loss of relevant data in translation and allows the software to perform more successfully within a broader toolkit. Understanding that this interoperability is critical for a functioning toolkit, it is clear why plugins developed for VPLs excel as members of toolkits. Such plugins take this concept to the extreme by allowing any relevant data to be connected/exported from one plugin to another. Of course, a plugin that outputs standardized data types is likely to be more successful at achieving this interoperability and this brings us to the next guideline in the list.

## Use Standardized Open Formats for Data Transfer

While interoperability with existing major software is critical for any toolkit, a good member of a kit also anticipates its compatibility with possible future extensions of itself and insertions of custom data at different times during its use. For this reason, the way in which data is passed between the different utilities of a tool can be just as important as the tool's overall ability to export to other platforms. This is particularly relevant as many programs use compiled proprietary file formats that are only readable by computers and cannot be easily translated to human-readable data, such as text, numbers, and geometry. While these compiled formats have some important uses in compressing data, their use in proprietary schema can severely limit a tool's ability to be extended and to "talk" to other tools. As such, the use of open text-readable standards like ASCII and UTF-8 greatly increases a software's usefulness within a toolkit. Furthermore, the use of standard file types for storing data, like CSV for tabular data or JSON/XML for object-oriented data, also helps enable cross-compatibility. Finally, VPL components that pass standardized text-readable streams of data between

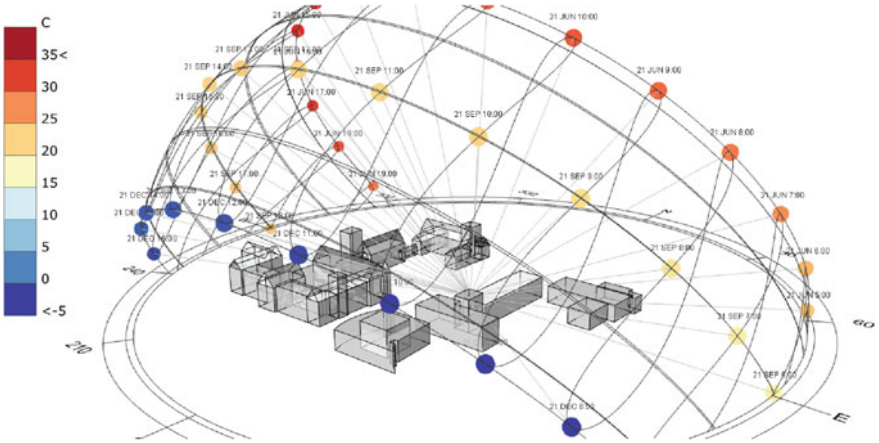


**Fig. 2** An example of annual hourly data usage in a Ladybug visualization: the suns of a sunpath are colored with hourly temperatures, denoting which parts of the sky should be blocked/shaded for thermal comfort

their components will more easily facilitate integration with other plugins and data sources. Perhaps the best example of this within Ladybug Tools is the structure used for annual hourly data, which can originate from several sources including downloaded climate data and annual building energy simulation results. Nearly all Ladybug visualization components can accept this hourly data as an input (Fig. 2) and both the standardization of this data across the plugin and the human-readable format of this data are critical to the success of Ladybug Tools. Specifically, this annual hourly data is derived from the standard format of .epw climate data and consists of a text header followed by numerical values for every hour of a year. This simple format enables both easy math operations to be performed on the numerical data while also providing text instructions for the specific components that make use of it. The fact that this data is human-readable also means that, if a user has hourly data coming from any other source outside of the plugin (like another piece of software or recorded empirical data), this can be directly input into Ladybug to visualize and analyze it. Other examples of standardized, human-readable formats in Ladybug Tools include text formats borrowed from its underlying simulations engines, like the Radiance standard for daylight materials and the EnergyPlus standard for full-building energy materials (Fig. 3).

## Modularize the Tool

The success achieved through the use of standard, text-readable formats initially depends on a tool being modularized into discrete elements that can pass this standardized data back and forth. The more modularized that a tool is, the more locations that exist for people to input/export custom data, build extensions on top of the tool, and connect it to other software. From this principle, we can understand that VPL

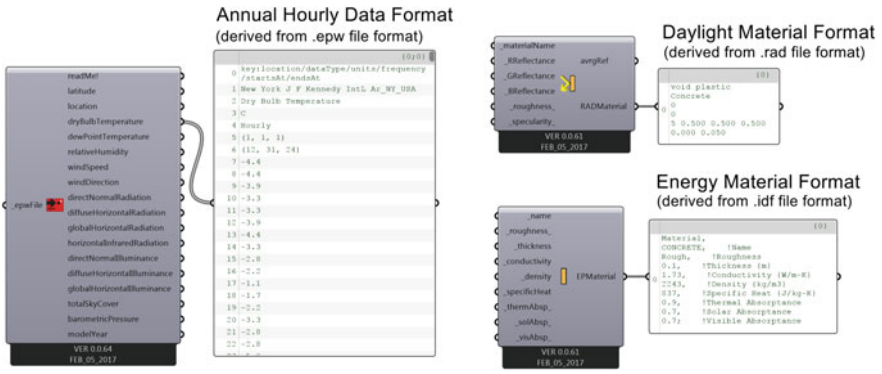


**Fig. 3** Standardized, human-readable data formats in Ladybug Tools

plugins will be more successful at integrating into toolkits if they break down their functions into more and more components or nodes. This is something that Ladybug Tools takes to heart since it is very rare to run an entire study with a single component. For a daylight simulation alone, one has separate components for geometry, materials, sky types, "recipes" (or simulation instructions), and result-processing (Fig. 4). This modularization ultimately allows for a much higher degree of customization and potential integration with other tools than would be possible if these processes were wrapped in a single component. It is also important to highlight that software does not necessarily have to exist in the form of VPL components in order for it to be modularized. The vast majority of software in the world achieves a modularization by breaking down all its capabilities into a well-documented Application Programming Interface (API) or Software Development Kit (SDK). The most "toolkit-like" of these APIs make use of a principle known as object-oriented programming, which essentially divides the functions of software into several different "objects," each with properties that can be set and operations that can be performed on it (Kindler and Kriv 2011). These "objects" can refer to anything and, as an example, the Rhinoceros CAD

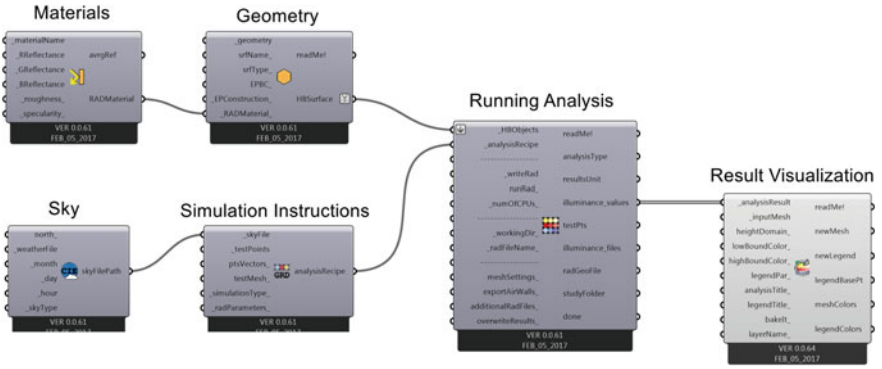


**Fig. 4** Modularization of daylight simulation in Ladybug Tools

software (McNeel 2017) includes several object types that one might readily recognize (like points, curves and surfaces) as well as objects that are less obvious (like an object for the viewport or an object for document settings). The more modularized, object-oriented, and well-documented such APIs are, the easier it is to build extensions off the software and connect it to other tools. Accordingly, VPL plugins that maximize their number of components and APIs that break down software into many objects tend to be more successful at operating within toolkit schemas. Given this principle, it is important to recognize that, the more components or objects that a tool is broken into, the steeper the learning curve is to mastering the software. For this reason, there is need for one final principle of the toolkit.

## Make It Easy to Start but Impossible to Master

Much like the instruments of any craftsperson, software toolkits are most successful when there is an art to mastering them. However, if such kits are too difficult to use from the start, new entrants can feel discouraged and will find it hard to engage. For this reason, the most successful toolkits follow a philosophy that was perhaps best summarized by the founder of Atari when describing their most popular video games—they are "simple to learn but impossible to master" (Bogost 2009). Following this mantra, the intense modularization, customizability, and exposing of options within a toolkit must be balanced with plenty of defaults for these options. Within Ladybug Tools, this manifests itself in the form of components that have large numbers of inputs but only a small number of them that are actually required to run the component. For example, the Ladybug sunpath has over 15 inputs, which allow for a high degree of customization, yet only a single input (the location) is necessary to produce the familiar solar graphic (Fig. 5).

This large number of defaulted inputs along with a visual standard to communicate which inputs are required or defaulted (using dashes _before or after_ input names) helps new users of Ladybug Tools navigate the capabilities of the software. In addition to default values, having a low number of required components for a given operation will further make a toolkit "easier to start." Within Ladybug Tools, this is best illustrated by the fact that only three components are necessary to run a full-building energy simulation. Yet, users can engage with over 100 other components in order to add meaningful aspects to this energy simulation (like window geometries, wall/window constructions, and energy-saving strategies like natural ventilation, shade, and heating/cooling efficiency upgrades). Structuring software like this enables new entrants to rapidly arrive at a tangible results and, with an understanding that these quickly-achieved results are far from perfect, users will be inspired to delve further into the toolkit. If done well, new entrants will find themselves quickly advancing through the kit and educating themselves as they go. Together, this forms a community of masters, new entrants, and many in between, who can help each other reach deeper into the kit through online discussion forums (Fig. 6). Eventually, masterful users may find a route all of the way to the core functions of the software and it is for this reason that many of the software packages that are most devoted to the notion of a toolkit are also

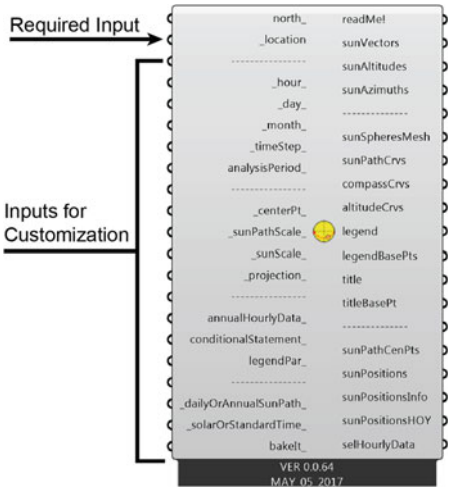


**Fig. 5** One required input at over fifteen customizable inputs on the Ladybug Tools sunpath

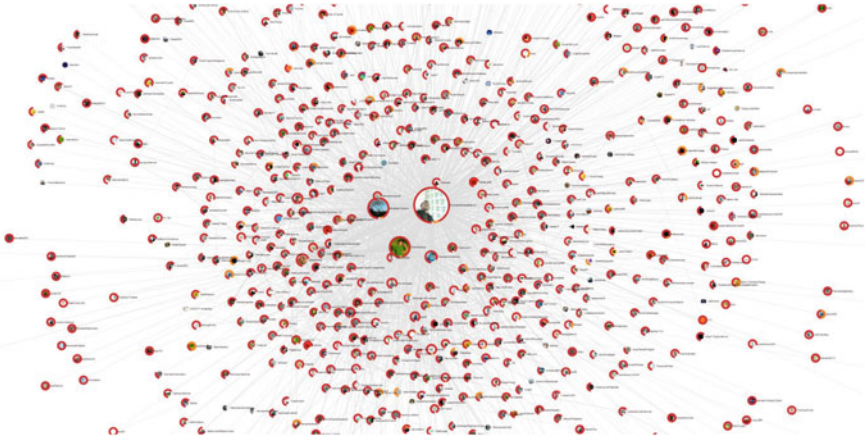

**Fig. 6** Snapshot of activity on the Ladybug Tools community forum (2017)

open source. Allowing everyone to view a tool's source code is an invitation for people to master it and, while not all software needs to be open source to participate in a toolkit, one can typically use this to identify software that intends to be participate as such.

By following the principles listed above, software can better meet the needs of today's architectural practice by enabling BOTH the flexibility to engage different iterative studies AND the integration that is needed to make coherent, coordinated designs. When we look at many of our buildings today, they increasingly resemble complicated collections of separate systems—structure, envelope, air conditioning, fire protection, electrical, interior furnishings, etc. In this separation of systems, we increasingly overlook opportunities for synergies between them, such as when part of the envelope can act as a structural system. This separation can also lead to misinformed decisions, such as removing exterior shade to save construction costs only to pay for it in a larger air conditioning system that can remove the higher solar gain. These missed opportunities are a direct result of our design thinking that is shaped by poorly integrated tools. If we are to have elegant, coherent building designs in this contemporary era, we must first address these underlying issues in our design workflows. By moving toward software toolkits instead of disconnected or centralized tools, our vision of elegant designs can more quickly become the reality around us.

## References

Autodesk: Dynamo. http://www.dynamobim.org (2017)

Bogost, I.: Persuasive Games: Familiarity, Habituation, and Catchiness. Gamasutra (2009)

Kindler, E., Krivy, I.: Object-oriented simulation of systems with sophisticated control. Int. J. General Syst. 313–343 (2011)

McNeel, R., & Associates: Grasshopper. Algorithmic modeling for Rhino. http://www.grasshopper3d.com/ (2017)

Moe, K.: Convergence: An Architectural Agenda for Energy. Routledge (2013)

National Renewable Energy Laboratory (NREL): OpenStudio. https://www.openstudio.net/ (2017)

Negendahl, K.: Building performance simulation in early design stage: an introduction to integrated dynamic models. Autom. Constr. **54**, 39–53 (2015)

Piker, D.: Kangaroo. http://kangaroo3d.com/ (2017)

Sadeghipour Roudsari, M., Mackey, C.: Ladybug tools. http://www.ladybug.tools/ (2017)

Tedeschi, A.: AAD Algorithms-Aided Design. Parametric Strategies Using Grasshopper. Edizioni Le Penseur (2014)

Vierlinger, R.: Octopus. http://www.food4rhino.com/app/octopus/ (2017)

Weber, S.: The Success of Open Source. Harvard University Press (2005)

Williamson, A.: An evening with Google's Marissa Mayer (2005)