Build Software Like Houses

Mel Conway

Executive Summary

The Subject. This paper is about the evolution of a vital contemporary technology: the technology of digital systems that interact with their users through graphical user interfaces (GUI) that both display output to the user and that accept user input. These are called "GUI tools" in the paper. This technology is everywhere–the GUI tools category comprises, for example, smart phones, personal computers, and many public kiosks–and many people depend on these tools for their livelihood, well being, and recreation.

The Problem. The problem that this paper addresses is that the great majority of people who interact with GUI tools do not have the slightest idea about what goes on inside them, and when something goes wrong, as it often can, somebody has to be called to help. Thus, to keep things running we must impose a class of elite intermediaries between many working people and the tools on which they depend. This situation is often unsettling, humiliating, frustrating, and disempowering–in sum, not good. It is also costly.

The Goal. This alienation between users and a vital technology on which they depend must be eliminated.

The Approach. In the paper I take a historical look at three vital technologies that have been with us for a long time—arithmetic in commerce, written communication between people, and the calendar in agriculture—and I notice that these technologies started out similarly to the way we are now with GUI tools, essentially requiring a class of elites or priests to administer, but over time these technologies successfully evolved from being obscure to being commonplace, to the extent that today all three technologies are taught to primary school children.

GUI tools are going to be with us for a long time, and it is important that they also go through the same evolution from obscure to commonplace that arithmetic, writing, and the calendar did, so that all users of GUI tools can use them with confidence. The question that I ask and then answer is: what will it take for this evolution from obscure to commonplace to happen? The answer I offer comes from observing how the three historical technologies—especially arithmetic—have been simplified for teaching to school children: take advantage of the hard-won hand-eye skills that each of us has spent

conway.mel@gmail.com

Revision date: 2/20/2009

years practicing as children. In other words, make it concrete and put it through the hands. I have done this for GUI tools.

The Solution. I have created a new, more humane, concrete conceptual model and a visual assembly method for GUI tools that will help any learner relate to the technology more like carpentry or plumbing than software. Furthermore, the concrete conceptual model and visual assembly method are applicable to the entire class of GUI tools. Today's GUI tools are not built according to this conceptual model, but they could be, and many would have more transparent implementations if they were.

The Question. We are now in a position to take a large step in the evolution of the technology of GUI tools from obscure to commonplace, just as arithmetic, writing, and the calendar have so evolved, with large social benefits. The reason is the appearance of a more humane conceptual model for GUI-tool internals. This new conceptual model is by its nature a disruptive technology. Where is the application niche in which this disruptive technology can become established? I present a few candidates, the most evident being in primary- and middle-school education. Not satisfied with existing answers to this question, I am eager to sit down with people interested in this issue to plan the best strategy for moving forward.

Contents of the Paper

conway.mel@gmail.com

Prelude to the Thesis

Digital Technology Connects to Our Hand-eye Skills

For thousands of generations men and women, and their genetic predecessors, have been securing and preparing their food, building their shelters, and otherwise modifying the world to support their existence, using their bare hands, sometimes augmented by simple hand tools.

The hand-eye learning style is built into us

To witness the development of a human child in the first few years of life is to realize the immense importance of the skills that intricately coordinate the hands and eyes. We are born without these skills but with a tremendous potential to acquire them, which we then obtain after birth through literally years of practice. Our species has evolved with a vital ***hand-eye subsystem*** in place to support both the *learning* of the hand-eye skills and their *employment*.

Our technologies have been evolving with us since prehistory

The development of our species began to accelerate when we started employing agencies outside our bodies to amplify our muscles, first domesticated animals, and later machines. We have acquired many such technologies over our history.

We don't use our technologies directly; we use *tools*. A tool is an invention that harnesses one or more technologies to amplify one or more human skills.

Something new is on the scene: digital technology

Now, in the last half century many existing technologies have fused and morphed into something that is so potent and strange we are tempted to call it a new technology. I call it *digital technology*. In some ways it is a higher order technology because of the way it interconnects, controls, and unifies many of our existing technologies.

The dominant tool of digital technology is the "*GUI* tool"*. **A GUI tool is a digital tool with a bidirectional interface to the tool user's hand-eye subsystem.** Examples: a word processor on a computer, a "smart" mobile telephone, a video game, an airline check-in kiosk.

> **This paper is about making the technology of GUI tools understandable to a large portion of our population.**

Technological Advance Alienates Its Users

It seems that every advance of a technology further removes the typical user of its tools from understanding how the tools work and from recovering from their failures. My favorite example is sound recording. Edison's development of the wax cylinder evolved into a mass consumer product, the 78 RPM record. Before the mechanical gramophone became electrified, its function and its connection to the ear was intuitively understandable by its users and, if part of it broke, its function could be largely recovered by the application of human ingenuity. If the needle broke, use a toothpick. If the spring broke, push the turntable around by hand. Even after the "morning glory" acoustic horn was replaced by an

---

* "GUI" (pronounced the same as *gooey*) is an acronym for *Graphical User Interface.* The term refers to the display screen of the prototypical GUI tool: the personal computer after the mid-1980s.

conway.mel@gmail.com

Revision date: 2/20/2009

electronic amplifier and loudspeaker, you could recover from failure by reverting to the mechanical model.

The replacement of the 78 RPM record by the 33 1/3 RPM LP began the descent into alienation. The grooves were so fine the toothpick trick no longer worked. Shortly after the LP record came magnetic tape; now the method of recording the music was invisible. But at least the sound waves were there–somewhere. The CD changed that; the sound waves became long strings of numbers, and even seeing the spots that carried these numbers required a powerful magnifier. Well, at least the numbers still stood for the sound waves, even though we couldn't find them. Then came mp3 compression and even that connection to our ear-brain went away. The iPod® is worlds away from the gramophone in elegance and power, but which will you choose when your life depends on keeping it running?

The story of movies is similar. Our technology has evolved from flip books to streams of bits that we can't even store and call our own.

Long-term Technologies Move Toward Widespread Understanding

What's wrong with the fact that technological advance separates people from the technologies they need? It's the disempowerment–our dependency on elite intermediaries between us and our technologies.

Arthur C. Clarke famously stated: "Any sufficiently advanced technology is indistinguishable from magic." Magic is beyond our understanding and therefore beyond rational control. For many of us, our livelihoods and security depend on magic of this sort. Because our control over this magic is at best tenuous, our dependency on advanced technology–and on the elites who manage it for us–is accompanied by a pervasive uneasiness.

But the news here is encouraging, if your time perspective is long enough. There is nothing new about the tension between technological advance and alienation of the masses. Indeed, one way we can think of human progress is in terms of the migration of the mastery of technologies from elites to commoners.

conway.mel@gmail.com

The boundary between the understanding elite and the dependent masses shifts, and understanding becomes universal

If a technology is important enough to last for a very long time, the fraction of the population that understands it may start with a small cadre of elites but it will increase until it becomes almost the entire population. As this happens, teaching about the technology will migrate from professional school to primary school. Consider:

- Arithmetic in commerce
- Written communication between people
- Use of the calendar in agriculture

Widened understanding is enabled by tools that are well matched to our hand-eye skills

The invention of easy-to-use tools contributes to the migration of technologies toward universal understanding. A tool that helps to make a technology universally understandable will necessarily be well matched to our hand-eye skills. Examples:

- Arithmetic became more accessible when we mapped numbers onto our hands. Then almost every person came equipped with counting tools. After that there was room for invention of the abacus as an amplifier of these counting tools.
- Writing became more widely accessible when the pen replaced the hammer and chisel. (In 1962 Douglas Engelbart, inventor of the computer mouse, observed that if you couldn't build a pencil smaller and lighter than a brick you wouldn't be able to teach writing to children.)
- The power of the rectangular grid as a tool for organizing thought gives us insight into how our eye-brains work. For example, the seven-column monthly calendar simplifies for us the unification of the (otherwise incommensurate) seven-day week and the 28-, 29-, 30-, or 31-day month. Many business consultants use 2-by-2 grids for simplifying choice-making situations. One of my favorite examples of the 2-by-2 grid as a conceptual tool: place four U.S. presidents in the following grid.

	Smart	**Stupid**
Active		
Passive		

- Indeed, the rectangular grid is the conceptual model of the groundbreaking GUI tool–the computer spreadsheet embodied as VisiCalc, 1-2-3, and Excel–that made digital technology useful to millions of people.

The Thesis

We have now laid the foundation for the question to be addressed by this paper:

> **What will it take for the majority of the population to understand the technology of GUI tools?**

This question is important because digital technology is the dominant technology of our era and we increasingly participate in our society and economy through the use of GUI tools. Our dependency on elite intermediaries for the use of such a key technology cannot be a social good.

My answer to this question is analogous to the way arithmetic has been made accessible to elementary-school children: *make it concrete and manually manipulatable*. In this section I describe in three steps a way to think about, and actually build, GUI tools using hand-eye skills, analogously to the way an amateur carpenter might build a dog house using hand-eye skills and a few hand tools. Hence the title of this paper: "Build Software Like Houses".

Widespread Understanding Requires Hand-eye Construction

Consider this task. You are given two pieces of wood, a hammer, and a nail, and asked to drive the nail into both pieces so they are fastened together. Here are two conceivable methods for performing the task.

1. Pick up the hammer and drive the nail into the two pieces of wood.
2. You have a robot that inputs nail-driving programs and executes them. Write a program that will be understandable by this robot, and give it to the robot.

Of course the choice is a no-brainer. But the example is not frivolous because the way we build GUI tools today is more like method 2 (programming) than method 1 (hand-eye construction). The more you think about method 2 the harder it seems to get. What language do you use to communicate with the robot? How do you tell the robot where to find the wood and the nail? How do you tell the robot how to deal with contingencies such as a bent nail or a hard knothole? Clearly, our hard-won hand-eye skills deal with a lot of questions implicitly in method 1 that we have to deal with explicitly if we choose method 2.

Is there something essentially different between programming and hand-eye construction? If so, what is it? I have identified four essential attributes of a hand-eye construction process: *unity, immediacy, continuity, and interactivity*; I describe these in Appendix 1.

The first step in the development of the thesis is this assertion:

> **A necessary condition for GUI tools to be explainable in elementary school (that is, to be widely understandable) is that they be constructible using a hand-eye process**.

A Concrete Metaphor for All GUI Tools Exists

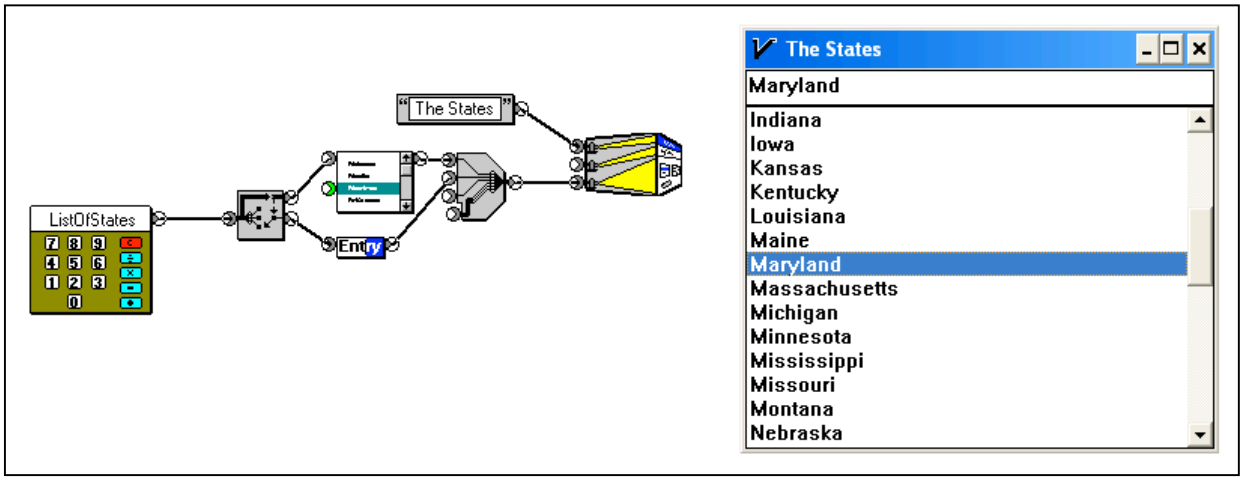
In this second step I present an alternative metaphor for the internal workings of GUI tools that is close enough to being concrete that it can be visualized and talked about by elementary-school children, given good instruction. In the third part of the thesis I shall describe a hand-eye tool for building GUI tools that conform to this metaphor.

Here are the parts of the metaphor.

- Lumps of data flow through pipes. Such a lump might be, for example, all the information about one person in a "card" of a computer's address book.
- The lumps flow from left to right in a plumbing network, some originating at files or databases at the left end of the network.
- Some lumps of data end up at the right end of the network, showing up on the user interface.
- On its way from its source toward the user interface the typical lump of data passes through one or more "transformers" that split up, combine, or otherwise change the form of, these data lumps. For example, several address cards might be combined into an address book, or the parts of one card might be separated into name, address, etc. The transformers have "connectors" that connect to the pipes. The data lumps flow into "sink" connectors on the left side of each transformer and they flow out of "source" connectors on its right side.
- The user interface is at the right end of the plumbing network. The user is to the right of the user interface, looking at it. The user interface can be thought of as a back-projection screen, and there are special transformers that cause data lumps to show up on the user interface; these "user-interface transformers" can be thought of as physically "projecting" the lump of data entering it onto its part of the screen.

I have worked with this conceptual model for years, and I have become convinced that it is a generally applicable model for GUI tools, and that there is a reasonably-sized basic set of transformers that makes it so.

The following figure shows an example of the user interface of a GUI tool (the user interface is on the right of the figure) and the plumbing network (on the left of the figure) that makes the GUI tool work. (In this example the GUI tool on the right is a Microsoft® Windows® program.) The function of this tool is simple: the user selects one name in the list of state names shown (in this case, the user clicks in the list box), and the name of the selected state shows up on the line just above the list box. This example is trivial, but it shows all the elements of the plumbing network described above, and it shows some of the implicit behaviors of transformers that give the model its power.



A Hand-eye Construction Tool That Builds GUI Tools Exists

The two images in the above figure are two parts of an actual screen shot of a computer screen on which the user interfaces of the following two distinct programs are running concurrently (extraneous material has been edited out).

1. The Windows program whose application window is shown at the right. The application's internal structure and operation is not conventional but is in fact the plumbing diagram that is shown at the left.

2. A visual drag-and-drop plumbing-design tool that built this application. Projected on the user interface of this plumbing-design tool is the network diagram shown at the left of the figure. It is important to stress that this design tool is not a translator; it is a what-you-see-is-what-you-get assembler of networks. *The program running at the right has the internal structure shown at the left.*[*]

Both programs are running concurrently. It is possible in the plumbing-design tool to inspect the data lumps in the pipes in real time, and one can watch them change as the user clicks on the list in the right-hand window. Furthermore, one can modify the plumbing network (without stopping the application) by removing or adding transformers and pipes, and the behavior and appearance of the application on the right can be seen keeping up with its definition on the left.

The paragraph immediately above this one characterizes the plumbing-design tool as a hand-eye tool that meets the necessary condition stated in the first step of the thesis development. Here, then, is the thesis.

> **We now have the ability to take the next major step toward widespread understanding of the technology of GUI tools.**

[*] How it works is described in detail in US Patent 6,272,672.

The Way Forward

This section is a work in process

What areas of application can we use to implant this disruptive technology? Two candidates occur to me: software development and education. I present my thoughts on each below, more to solicit reaction than to be definitive.

Software Development

I have studied at length the potential impact of the technology described here on the development process for software-based GUI tools of all types and I believe that, if the technology were to be widely adopted, it could profoundly simplify the development process and reduce costs. There are two main reasons for this impact: network effects of shared development of a growing body of reusable transformers and the entrance into the workforce of a large number of people who could not otherwise have become productive application developers. Additionally, for some classes of GUI software applications, the development process would become more informal and open.

My experience describing this opportunity to many technical audiences over more than a decade suggests that the uptake of this technology by the software development community is unlikely. There is a large investment in the existing ways of doing things that, for valid reasons, will resist a change in conceptual paradigms. Alternatively, the approach could be used just as a design methodology that would lead to better cost estimation and smoother development, but there is a lot of competition in this area that makes similar claims.

> There is one corner of software technology that is too new to be settled: development of applications that reside in the Internet "cloud". Appendix 2 contains a description that I have presented of a cloud-based application-development service.

Education

To teach the model described here in high school or college as a construction methodology might be pedagogically interesting and ultimately valuable to students, but people of high-school and college age are looking for knowledge that is directly transferable to employment. This means using available instruction time for teaching existing software development methods.

My conclusion is to go where the deterrents enumerated above do not exist, yet where there is a strong attraction to quick uptake of confidence and transferable skills: primary- and middle-school education.

I foresee teaching children to build GUI tools that will be interesting and educationally valuable to them—games, interacting robots, and social networking environments, for example. Having experienced powerful success the children will have confidence in their intuitions about the workings of GUI tools. This confidence then becomes an aptitude for learning existing software technology.

For an education project of this nature to be realized, the existing construction tool would be a prototype for development of a new

construction tool robust enough for the primary classroom environment; additionally, educational materials would have to be developed. These are substantial activities that would take time and resources and would therefore require a strong commitment to the outcome.

Where can we find a strong desire to empower school children with confident understanding of GUI technology? The global audience of the One Laptop Per Child project has demonstrated its attraction to the empowerment that easily learned computer technology grants young students. In the United States, however, the idea of engaging primary-school children in GUI technology does not yet exist on a large scale; this idea will grow as a political consensus grows that primary education is preparation for global economic life. Perhaps such a political consensus is beginning to develop at this moment in U.S. history.

* * *

The ideas presented above deal with doing better something that is being done now—more or less. What follows proposes doing something vital that is not being done at all.

As a public high-school teacher I learned that my students had no preparation for collaborative problem solving. Here is an except from my "Urban Teaching" blog.*

> Virtually all of my students arrive with no understanding of working in teams. They have neither expectations nor skills for working together in order to work more effectively. I have experimented extensively with attempting to create team problem-solving environments in my classes, and I have concluded that what is arguably the most important workplace skill that can be given to high school students, the ability to create and work together in teams, has been almost completely ignored.

The technology described in this paper offers a platform for teaching a 21st-century skill now ignored by our schools: cooperation on globally distributed construction projects. (Indeed, our locally focused education system has no way to teach—or even to discover the need for—such a globally focused skill.) The nature of the construction tool as a web application and the extensibility of the transformer library are enabling technologies for cooperative construction projects that can be carried out over the Internet.

I see an opportunity to develop a curriculum in geographically distributed software application development. The focus of the curriculum would be project management but the students would be building several things: skills in working cooperatively at a distance, software skills, and a valuable software application. An important part of curriculum development would be the definition of a deliverable application that will be both useful and important to the students, for example a tracking system for global temperature change.

---

* The essay "A Radical Proposal" at http://web.mac.com/melconway .

Appendix 1: The Four Elements of the Hand-eye Construction Process

Note: This discussion is about construction of software, but its conclusions are not restricted; the definition applies to all construction processes.

One important feature of spreadsheets is that they react immediately to changes made by the user. As humans with a long history of using manual tools, we take such a characteristic for granted. The process of pounding a nail through two pieces of wood is a directly interactive process. You don't map out and write down in advance a step-by-step program describing your nail-pounding plan. You hit the nail. You see what happens; that tells you how to hit the nail the next time. When it's driven home you stop hitting it. Almost all manual tools used by people in their work depend on a tight hand-eye feedback loop for their use, from the surgeon's scalpel to the potato peeler.

The conventional software construction process (the spreadsheet being a notable exception) isn't like that. Building a software application using most contemporary tools is more like baking a cake without a recipe. You decide how you want the cake to be, you use your considerable experience to guess at a recipe, and you write the recipe down. Then you spend an hour executing the recipe, i.e., putting together the batter and baking it. Not quite. Do it again. The process takes a lot of record keeping and patience. That's what most of today's programmers do.

One of the problems with arriving at a recipe this way is that the output (the cake) bears no physical or logical resemblance to the input (the recipe). Executing a recipe (i.e., baking a cake) creates something totally different from a recipe. So you have to combine guesswork and experience to decide how to change the input in order to fix the output. There is an inconvenience in this: you have to wait an hour to see whether your changes improved things. There is a more serious problem: it's not necessarily obvious how to convert your dissatisfaction with the output into a change to the input. It's not like pounding a nail where if the thing bends you see immediately how to straighten it.

We have concluded that, in order for building software to be far simpler than it is today, the construction process must have four attributes that will make building software much more like pounding nails. If the process has these four attributes we call it a *hand-eye* process.

1. **Unity.** There is only one thing being worked on, and it's *both* the input and the output. Nailing two boards together means transforming, in a sequence of steps, two boards and a nail into two boards nailed together.
2. **Immediacy.** When the builder swings the hammer, the nail moves and the eye-brain immediately understands the new state of the thing being worked on.
3. **Continuity.** Small actions produce small changes. Hit the nail harder and it will probably move more. This characteristic helps to tell you how hard to hit and when to stop.
4. **Interactivity.** Strategizing is part of the hand-eye feedback loop. How you hit the nail this time depends on what it did the last time you hit it. The sequence of steps required to do the whole job is not predetermined; each step determines the next.

Appendix 2: A Widely Usable Web Application Development Service

Use of the Service. This service enables its user-developers to build interactive Web-based applications capable enough to meet the needs of businesses.

Access to the Service. A user-developer can access the service from within any contemporary broadband-connected browser.

Graphical Development. The language with which the user-developer works is a concrete, visual "plumbing" metaphor. Simple applications are visual and easy to build. Complex applications are complex, but still visual and still possible to build.

The User-Developer's Application Model. The user-developer of this service creates and edits visual plumbing diagrams within a Web browser. For a potentially large population of user-developers the plumbing application model is more understandable than programming, yet generality, scalability, and application performance are not sacrificed. The plumbing network of an application can be hierarchically structured to limit visual clutter and enhance comprehension. The service enables encapsulation of plumbing diagrams to create new transformer definitions; these can then be added to the transformer library.

Location of the Application Server. By default the application the user-developer is building executes from the service's server. Whether the service also offers the option to export a finished application to the user-developer's own server is a business decision.

A Suggested Collaboration Model. The service offers "open source" transformers in public libraries. The service can choose to offer private transformer libraries to certain users.

A Suggested Public Project. The service can be the focus of a world-wide open-source project that develops a library of application transformers and business objects with which the service's user-developers can build enterprise-level applications.

Risk Assessment. The following table describes the estimated current technological risks in the development of the proposed service.

Demonstrated and documented	Considered feasible, not yet demonstrated	Needs to be demonstrated
How the user-developer's applications work. How the development tool works. How transformer library extensibility works. How the model is scalable and complexity can be hidden. (Full disclosure is in U.S. Patent 6,272,672. The author is the sole inventor. The patent has never been assigned or licensed.)	Determination of a practical set of primitive transformers—built by coding—from which all other transformers are built by encapsulating plumbing diagrams. (The estimated size of the primitive transformer library is 100 to 200.) A large-scale demonstration project.	Can a user-friendly drag-drop-style wiring tool be built to run over the Web with acceptable performance? The technology would be partitioned as follows. • The plumbing-diagram-structured application the user is building is in the service's server. • The user's browser presents an editable plumbing-diagram view of a portion of the application's structure.