An Application Development/Execution Platform for the Rest of Us

Mel Conway, March 13, 2020 @conways_law

This is about engaging almost everybody in the process of building real-world applications. It develops the rationale for, and will end with, a product concept.
0:1/2

Contents:

0:2/2

1:Thesis

If non-programmers are going to build real-world solutions
a simpler programming language isn't enough.

We must put non-programmers in a
collaborative situation
where we meet them where they are
and the environment empowers them by leveraging developers' skills.
1:1/2

So this is about creating a new

platform-based two-sided software market
in which almost everybody can participate with the skills they already have.
1:2/2

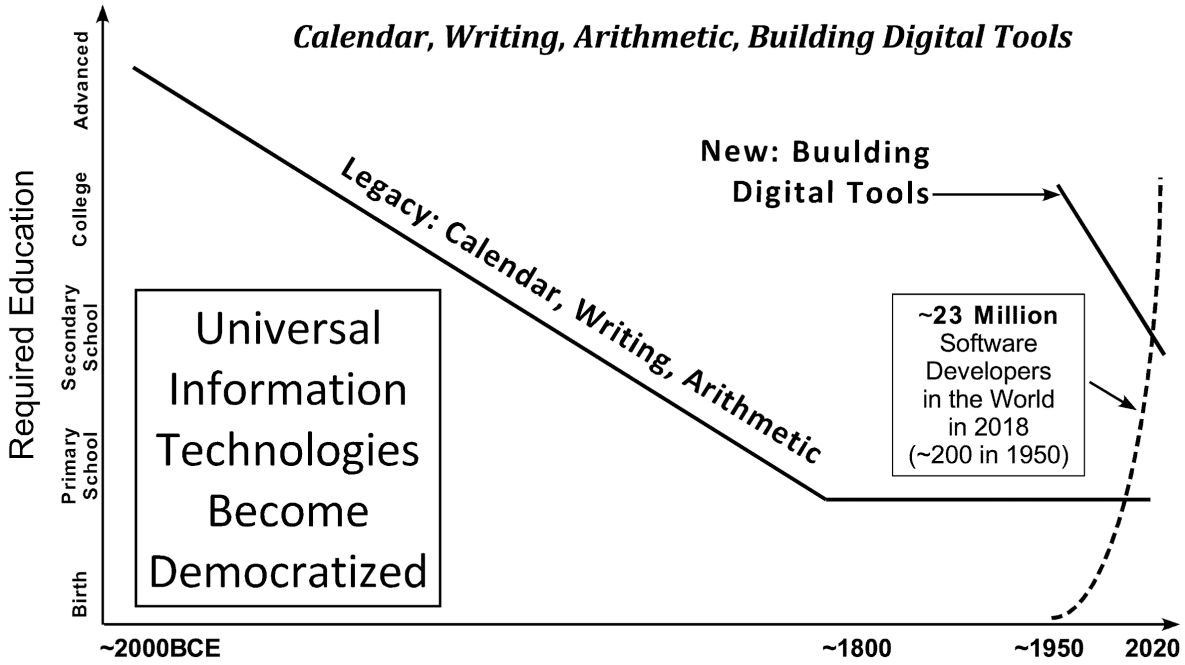
2:Historical Perspective

The software industry is in the middle of a monumental historic transformation. What we might not realize is that this transformation has been going on for 4000 years:

Democratization of culture-critical information technologies.
2:1/16

The first three legacy universal information technologies:
calendar, writing, and arithmetic
started off as the property of an elite priest class and, over approximately 4000 years, are now being taught in elementary schools around the world.
2:2/16

I believe that building digital tools is so central to our technological society that it’s going to join these three legacy technologies as the fourth universal information technology. This paper is about how that might happen.
2:3/16



This chart shows how the level of education required to use a technology has dropped over 4000 years. The vertical axis shows the level of education, from birth (not a joke, as you will see) to advanced education.
2:4/16

For the three legacy technologies the long solid lines show the democratization from high priest to elementary-school child. Now along comes a fourth, building digital tools, beginning around 1950, entering at the far right.
2:5/16

Note the dotted line at the right. It shows that over the last 70 years, the number of software developers has grown from about 200 in 1950 to over

23 million in 2018. That's an average growth rate of about 18% per year over 70 years.

Something big happened around 1950.
2:6/16

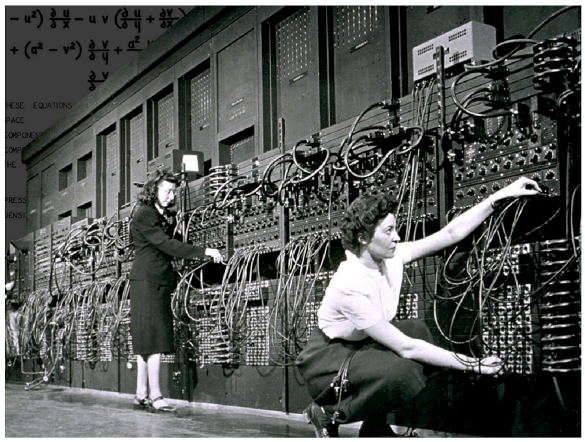
Meanwhile the educational requirement to build digital technology has dropped from college to early secondary or middle school for a lot of kids.
2:7/16

What happened to the legacy technologies:
1.Their conceptual models became simpler and more accessible to more people, and
2.Their media became more lightweight, portable, and easily reproducible.
2:8/16

Here's how it used to be. Mayan time was circular. The Mayan calendar was three concurrent cycles: 260 days (for ceremonial purposes), 365 days (for agriculture), and approximately 7900 years (essentially a linearization for counting indefinitely into the past or future).
2:9/16

You might recall that a few years ago there was a doomsday myth going around because all three cycles simultaneously came to their origin points in December 2012.
2:10/16

Programming Before and After 1950



ENIAC, ~1946

PC, ~1985

What happened around 1950 was a
step-function reduction
in the skill level required to program computers, hence a
step-function increase
in the number of people to whom programming became accessible.
2:11/16

A 1945 paper introduced the idea of representing a computer program as data in the same memory used for problem data, rather than as external plugboard wiring.

Another important change was in the weight of the embodiment of a program. (The Mayan calendar vs your phone.)
2:12/16

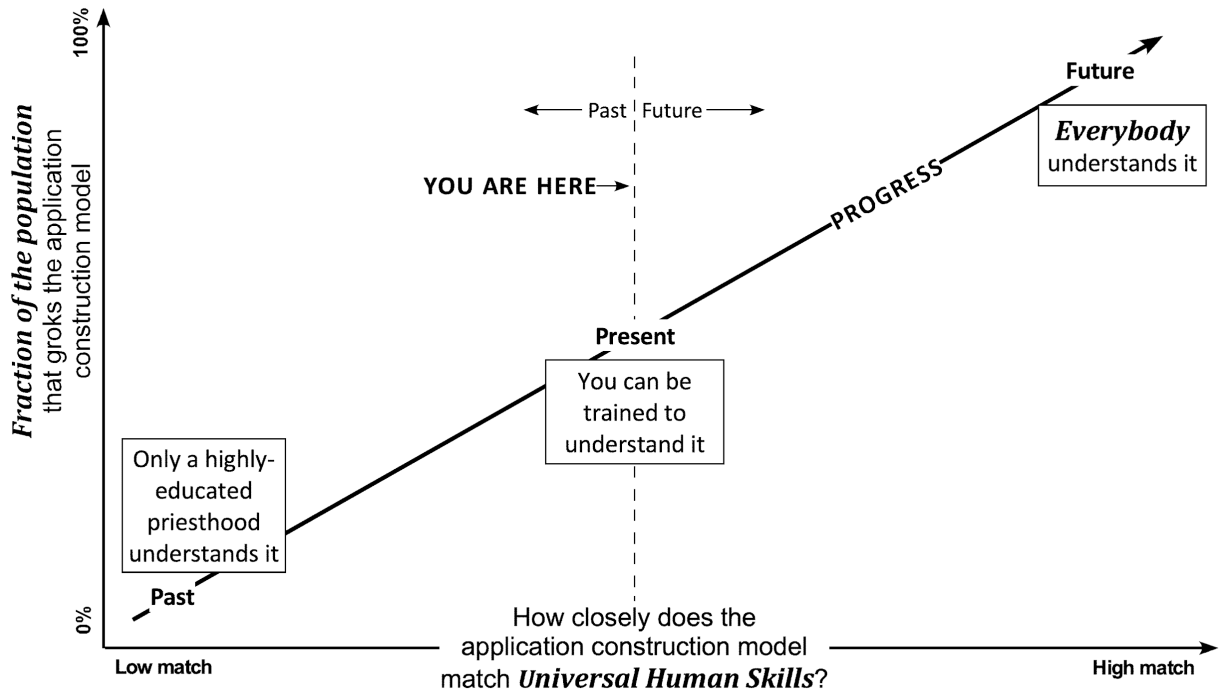
ENIAC programmers in 1946 had to know digital logic as well as mathematics. ENIAC program wiring could take days to set up and weeks

to complete test, and you had better be sure you had run all your data before you tore it all down for the next application.
2:13/16

And while you're wiring it up and tearing it down, the computer is sitting there unavailable.
Also, communication, training, and other delays were greatly reduced due to the lightweight medium, which is easily copied and transmitted.
2:14/16

The first commercial computers appeared in the early 1950s. Then, people like me in the universities were building assemblers and compilers. IBM introduced FORTRAN in 1957, which brought a lot of scientists and engineers into programming.
2:15/16

Meanwhile, Moore's law was continually driving down the costs of hardware. Desktop computers started in earnest in the early 1980s, greatly enlarging the market for applications.
2:16/16



3:The Model of Progress

This graph introduces the concept of *Universal Human Skills*. It says that the more your technology relies only on Universal Human Skills and doesn't require additional education, the more people will be able to use it.
3:1/16

The line suggests progress, from a past at the left where the model is not intuitive and only a few people get it, to the present in the middle, where you can be trained to understand it, to the top right where everybody gets it, maybe at some point in elementary school.
3:2/16

But before we go further, we have to answer what we mean specifically by “Universal Human Skills”.

What skills do all humans have in common that we can exploit in order to design a fully accessible model of how to build software?
3:3/16

According to Michael Merzenich of UC San Francisco, what all of our brains have in common is
plasticity.

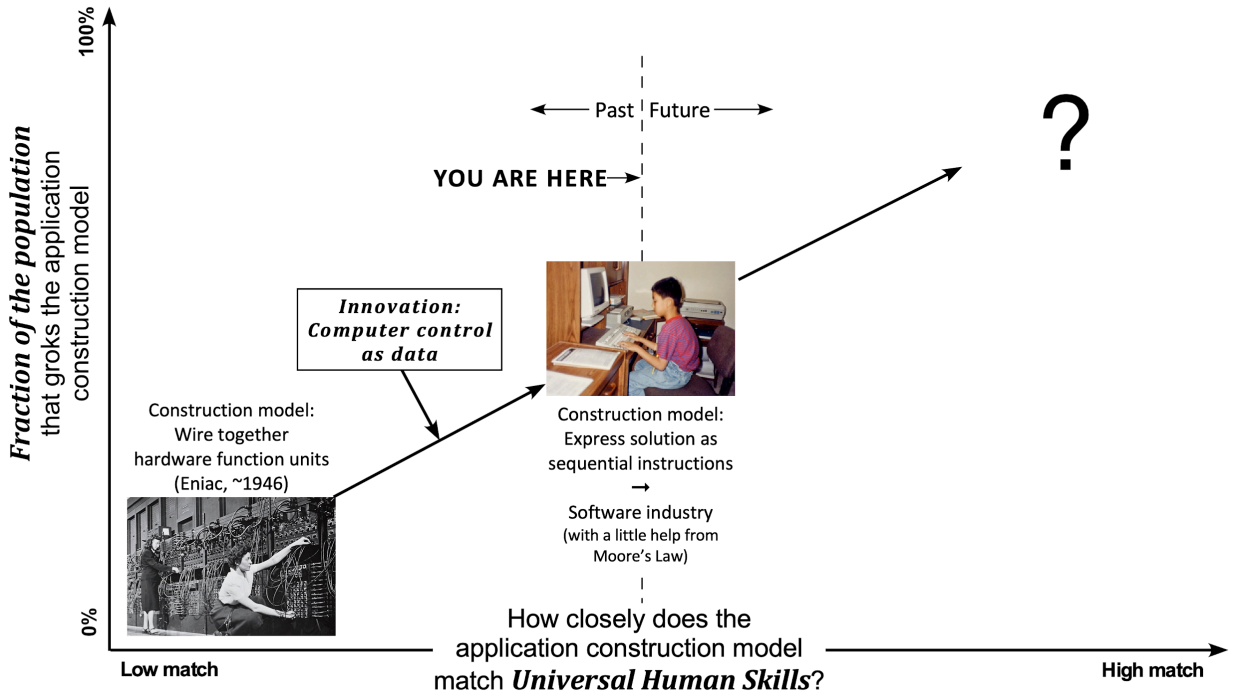
The first few years of the life of each human child are massively invested in this project:

Build My Brain.
https://www.ted.com/talks/michael_merzenich_growing_evidence_of_brain_plasticity
3:4/16

Nature builds into every infant a process of building its brain, first passively by responding to stimuli, and then actively, through relentlessly persistent practice, well beyond what we as adults normally can tolerate.
3:5/16

For most normal children this project focuses on hand-eye-brain coordination.

Therefore:
Universal Human Skills
means
hand-eye-brain coordination.
3:6/16



In this graphic the picture at the lower left shows ENIAC programming around 1946. The ENIAC was no slouch; it was built to compute artillery tables in World War 2 and was also used to study the feasibility of thermonuclear fusion.
3:7/16

Programming ENIAC required people trained in both mathematics and digital logic to wire up the functional units.

They were almost all women, by the way, because the men were in the military fighting World War 2.

Women were leaders in the early days of software.
3:8/16

The invention in 1945 of the stored-program computer changed everything. It eliminated the need to understand digital logic and opened up the creation of computer solutions directly to the scientists who had the problems, greatly simplifying the solution process.

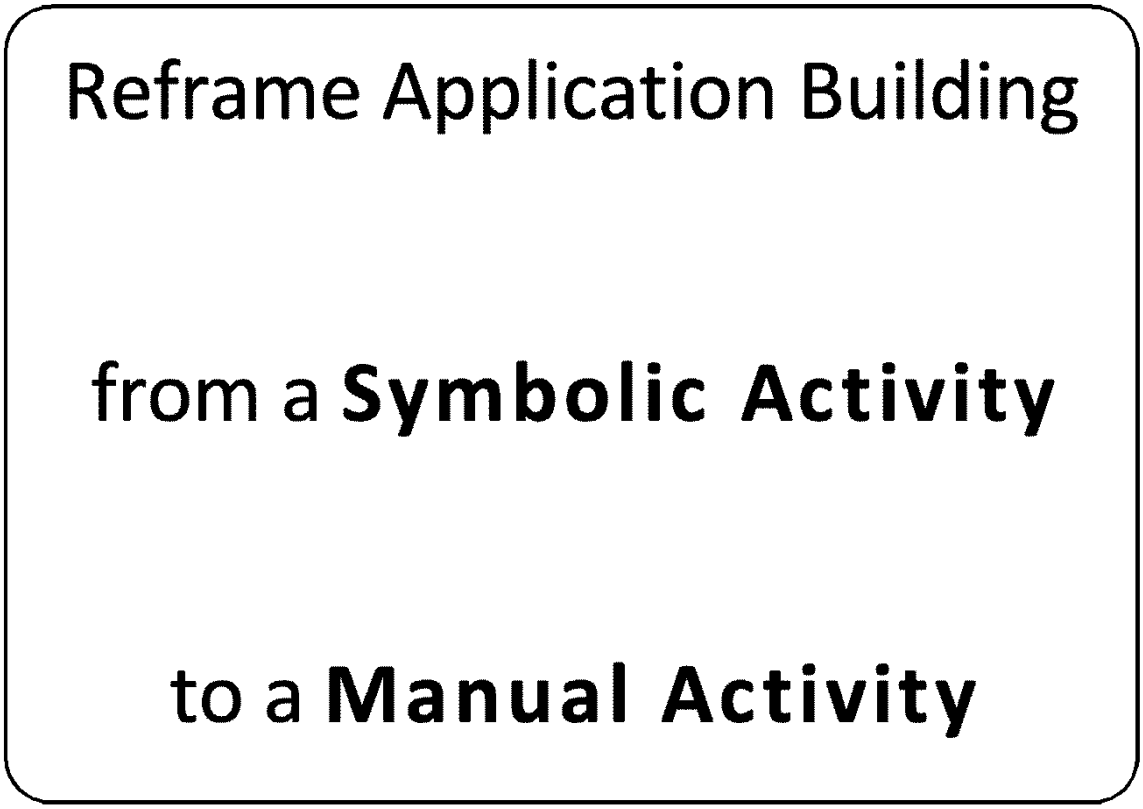
3:9/16

Also, programs were now data, and computers could create programs. High-level languages and their translators were born, simplifying the task and bringing in still more people.
3:10/16

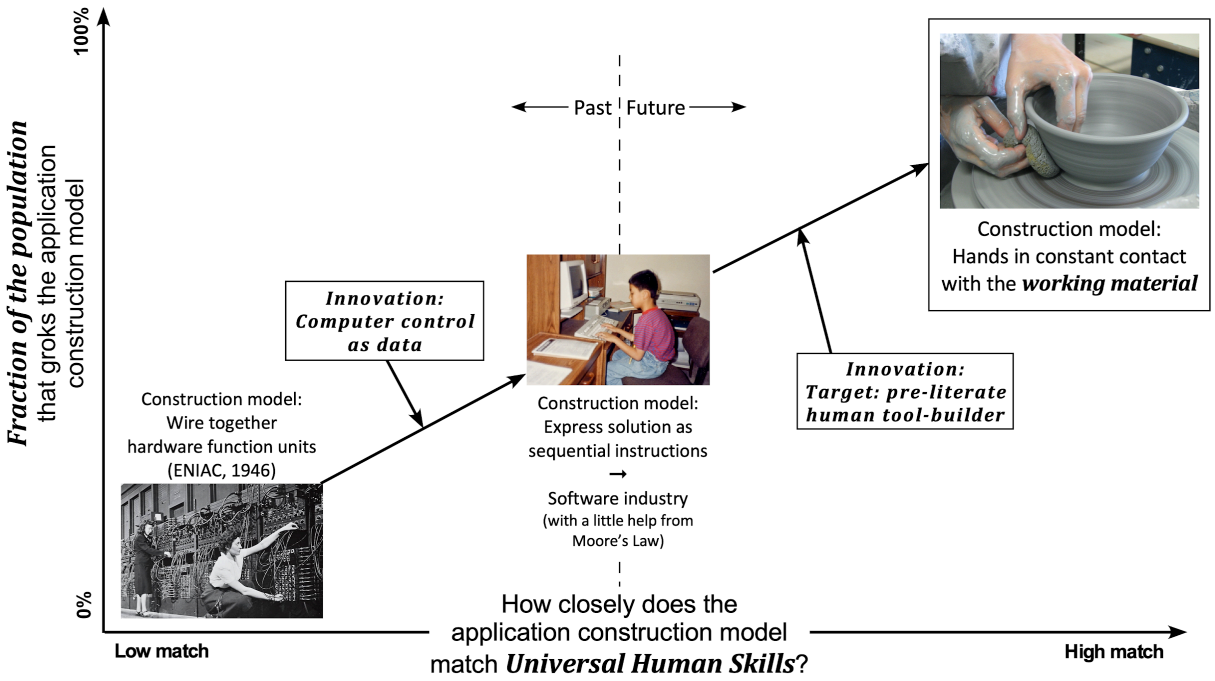
The resulting explosion in the number of programmer candidates created the software industry, and now we have over 20 million developers worldwide.
3:11/16

That question mark at the upper right, what does it tell us?

That the construction model must be accessible to almost everybody. It must be a high match to Universal Human Skills, that is, hand-eye-brain coordination.
3:12/16



This is the big idea. We must reframe application building from a symbolic activity to a manual activity.
What does that even mean? How do we do it?
And then, what do we even mean by a “programming language”, or by “programming” itself?
3:13/16



The potter's wheel at the top is a great model of hand-eye coordination in building something. There is immediate feedback involving the hands, the eyes, and the working material.
3:14/16

What is software "working material"?

The term suggests that the thing we're building evolves as a valid thing from the very beginning of its life. The bowl starts out as a lump of clay and has a continuous existence as the potter forms it.
3:15/16

What if we need to rethink the application's model as being continually valid and observable from the very beginning of its construction?

I’ve addressed these questions and have come up with a list of a dozen tool design principles, which I call the
Humane Dozen.
3:16/16

The Humane Dozen

Twelve Design Principles for ***Humane Tools***

Hands on the working material

- Immediate
- Continuous
- Interactive
- Transparent
- Inspectable
- Modifiable
- Robust

Minimum mental gear-shifting

- Unified
- Self-revealing
- Symmetrical
- Always on
- Alive with your data

4:Humane Tool Design

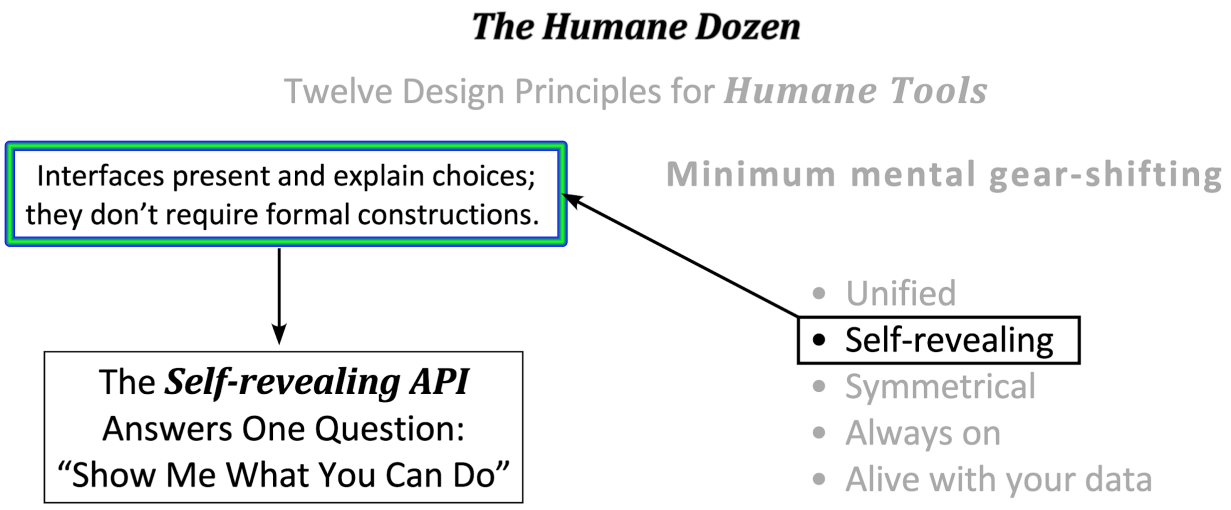
These twelve principles can't just be tacked onto existing development tools. They form a holistic, "Humane", approach to building and testing.
4:1/14

This three-page paper gives you one or two sentences for each principle.
http://melconway.com/Home/pdf/humanedozen.pdf
4:2/14

Here is one consequence that falls out of fully following the Humane Dozen that you might recognize in browser debuggers.
4:3/14

The tool and the application it's building (with live data) are in front of you as peers. Your next event can be in either one of them. If you change the application in the tool, its behavior immediately reflects the change. Just like turning a bowl on a potter's wheel.
4:4/14

If you try to do something illegal it refuses and tells you why, and even doing stupid things won't break anything. What has to happen for that to work?
4:5/14



Let's take just one principle, self-revealing, and describe how it might apply to one application of that principle: API design.

The self-revealing principle is quoted in the green box.
4:6/14

This is old stuff; it's how the Mac and Windows replaced the command line with dialogs and icons. They called it the "WIMP" interface, for Windows, Icons, Menus, and Pointer.
4:7/14

You saw with Mac and Windows how
lowering the barrier to entry greatly increased the population of users.

That's where we have to go.
4:8/14

Every self-revealing API is an
active object
that answers this one question:

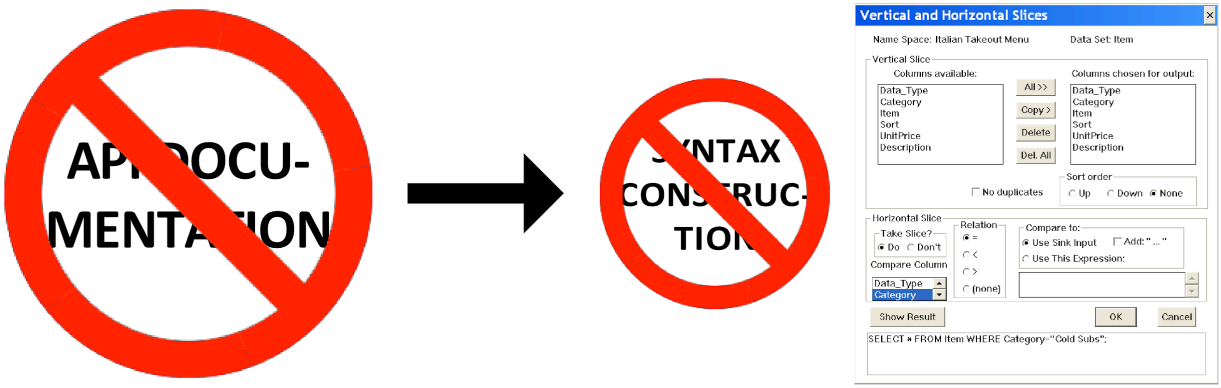
Show me what you can do.

So, even with this simple example, we've changed the tool concept.

We've replaced external API documentation by tools that interact with active APIs.
4:9/14

The ***Self-revealing API***
Replaces API Documentation by

Dialog-based Interactions Within the Development Tool



At the lower right is a dialog for building an SQL Select. It looks familiar. But it's not that obvious because of another principle: Alive With Your Data. That principle means that you're watching the result of your query as you build it, step by step.
4:10/14

The tool asks the API: Show me what you can do.

The API responds with a list of services that the tool lists, including help provided by each service if requested. Once the builder chooses a service

the service takes control

and presents whatever WIMP dialogs it needs.
4:11/14

It won’t be simple, so why try?

Because doing so can greatly enlarge the number of candidates for building applications. Empowering new classes of builders means

creating new markets and building new businesses.
4:12/14

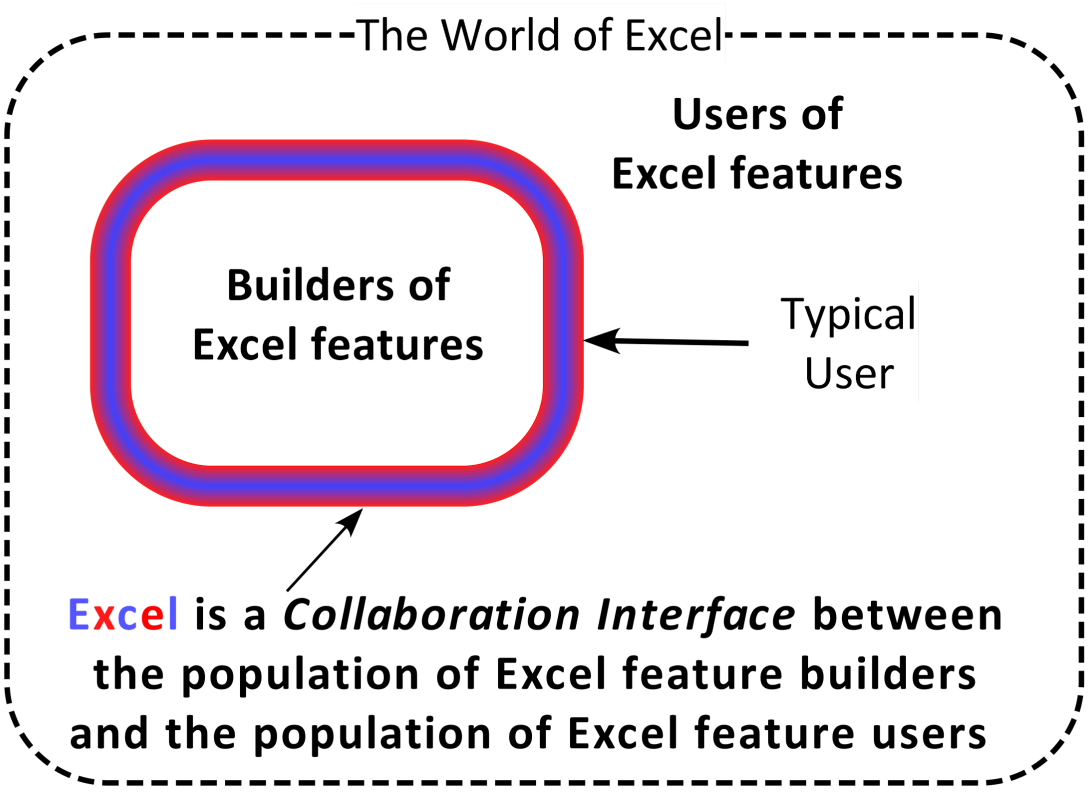
This is not about today’s “low-code” and “no-code” languages, whose vendors see them as productivity enhancements within the enterprise.

It’s about a holistic market-based construction concept that is addressed to the total population.
4:13/14

It’s research, but it’s not pie-in-the-sky. I’ve worked through many of the issues and have a working Humane proof of concept.

So what does a software market based on this design concept look like?
4:14/14

Key Idea 1:
Think Populations and
Collaboration Interfaces Between Populations



5:On Platforms

Let's consider Microsoft® Excel as an acknowledged and widely used approach to simple application building, but let's consider it not as a piece of technology but as an

interface between two populations of people.
5:1/20

Excel is more than a programming language. A big part of its power is its library of functions and presentations.

Excel is a platform

that enables non-programmers to reuse a portfolio of professionally-built functions and presentations in a humane development environment.
5:2/20

In the graphic everybody in the world of Excel is inside the outer rectangle, and inside that there is a colored rectangle containing the builders of Excel features.

Excel itself, the red and blue boundary, is a

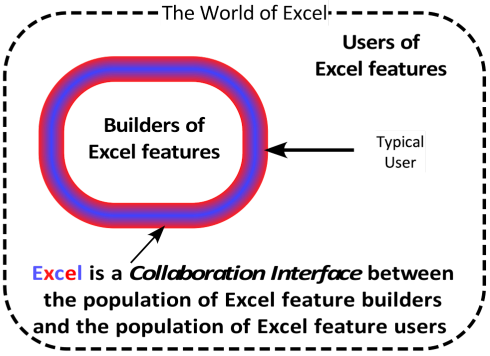
collaboration interface

between these two populations.
5:3/20

So Key Idea 1 is the notion of two populations with different skills and a collaboration interface between them.
5:4/20

Key Idea 2:
Think Mode of Collaboration

- **A**synchronous
  They work at different times, through the interface
- **A**symmetric
  They have non-communicating skills
- **C**onstruction
  They're building something



We're interested in these three key attributes of the collaboration interface.

Asynchronous.
The builders of Excel features and the users of Excel features work at different times, but they collaborate through the platform, which combines their work into something new.
5:5/20

Asymmetric.
The builders and users of Excel have very different skill sets. Some platforms with more matched populations, like telephones, don't have this property, but asymmetry of skills is a key idea to leveraging the skills of the builders in order to empower the users.
5:6/20

Construction.
Some platforms, railroads, for example, don't build new things as the result of their use. Some, like the potter's wheel and software development tools, do.
5:7/20

So What's a Platform (1 of 3)?

To a layperson: Something you put stuff on

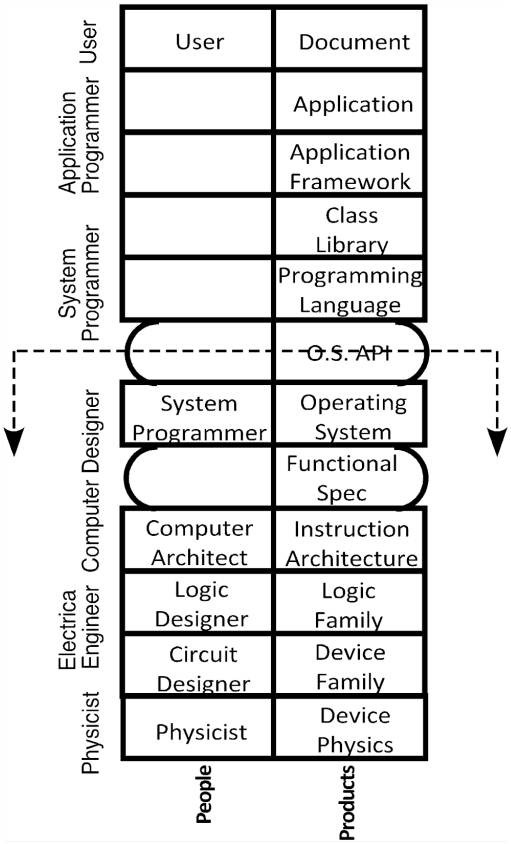
What's a platform? It turns out that it depends on whom you ask.

To the person on the street, a platform is something you put stuff on.
5:8/20

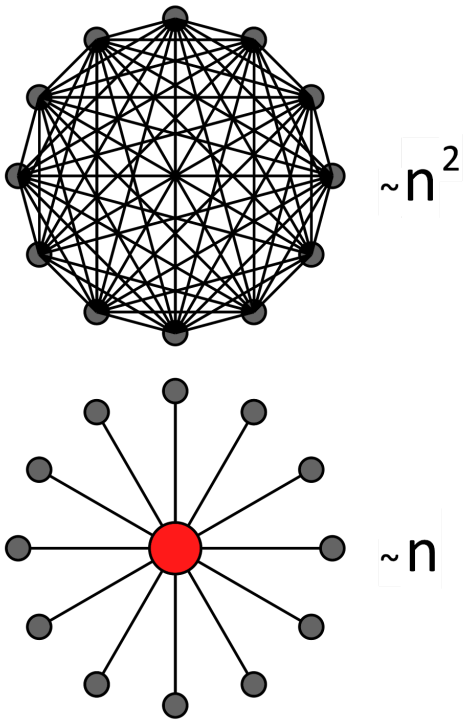
So What's a Platform (2 of 3)?

To an engineer:

The bottom of a technology stack



A hub that makes a point-to-point network into a star



I've seen two different usages of the word by software engineers, shown here.

In the example on the right, the number of connections in the star network grows linearly with the number of nodes, whereas in the point-to-point network it grows quadratically.

5:9/20

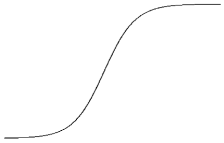
So What's a Platform (3 of 3)?

To an economist:

These are all platforms:

- **Railroads (especially 19th century)**
- **Phone system (esp. 20th century)**
- **VISA, MasterCard, Amex**
- **UBER, Lyft**
- **Airbnb**
- **Facebook, Twitter**
- **Alibaba, Amazon**

What makes them important:



Network Effects
Will make you or break you

Interface standards
Are everywhere

To an economist, though, a platform is a powerful economic phenomenon that connects two populations in what's called a two-sided market. You'll recognize the examples on the left; they're all associated with rapid market growth.
5:10/20

The two populations are sometimes called producers and consumers, but the suggested asymmetry is not an essential part of the definition.
5:11/20

Network effects
are key to market growth. The more people on one side of the platform, the more valuable the platform is to the people on the other side. In the early days of the telephone you wanted a phone service that connects you to the people you want to talk to.
5:12/20

The curve, called the “logistics curve” describes the early exponential growth (due to this positive feedback) and later market saturation. Once a platform with strong network effects gets started it’s really hard to dislodge it.
5:13/20

When you examine successful platforms you see a lot of standardization of interfaces

which further enhances network effects.
5:14/20

If you want to study platforms as economic phenomena I recommend the book “Matchmakers” by Evans and Schmalensee.

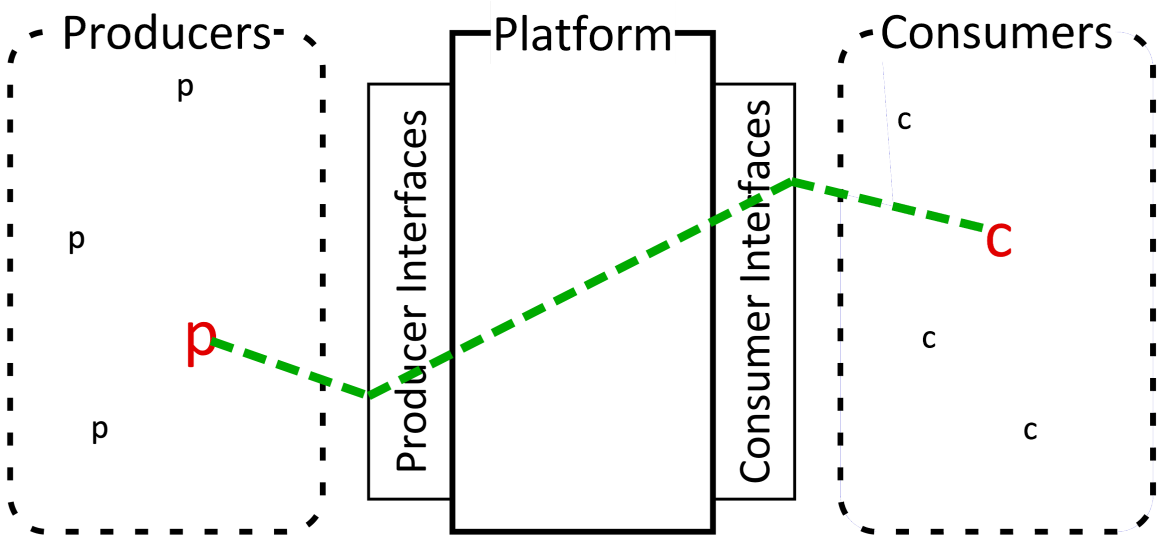
“First-mover advantage” is good-news/bad-news. The first mover has a lot to learn, and will make mistakes. It's often the second mover that captures the market.
5:15/20

A **Platform** is a Communication Channel

Between Two **Populations**

That Enables Mutually Beneficial **Transactions**

Between Their Respective **Members**



Here is my definition.
5:16/20

My favorite interface standards picture: the port of Shanghai.

From the ships and cranes in the distance to the containers and trucks in the foreground, to the barcodes on every container connecting it to a worldwide information system, count all the interface standards!
5:17/20

When intermodal containers were introduced in 1956, most cargoes were loaded and unloaded by hand by longshoremen. (Remember the movie “On the Waterfront”?) This introduction reduced the cost of loading a ship by a factor of 36, from over $5 a ton to 16 cents a ton.
5:18/20

Containerization also greatly reduced the time to load and unload ships, which reduced unproductive time in port.
Since that time the volume of international shipping has exploded.
5:19/20

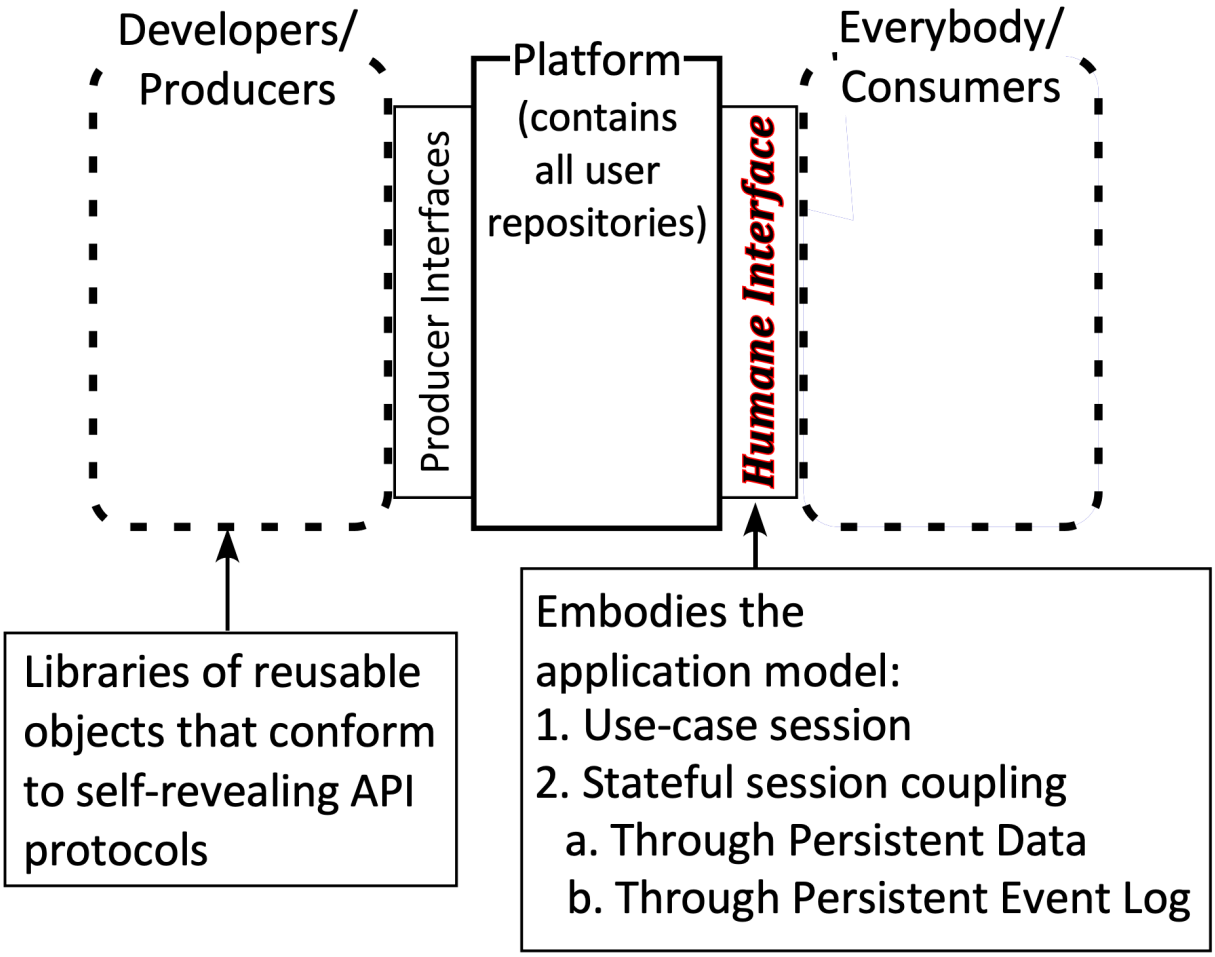
Comparing containers to longshoremen is analogous to comparing stored programs to bundles of wiring, in their effect on setup costs.

Notice software being framed here in a systems model that has parallels to other businesses, such as shipping.
5:20/20

6:Where the Work Stands

Now the pieces are in place to describe the goal: a

Humane
Asynchronous
Asymmetric
Construction Platform
for Stateful Business Applications.
6:1/17



This is what the cloud-based development/execution platform looks like. On the left are developers building reusable components and on the right is just about anybody assembling these components into applications and other reusable components.
6:2/17

The Consumer Interface on the right presents the application model to Everybody in Humane form.

What does it really look like?

I can answer by dividing the application model into two parts:
1.each use-case session, and
2.stateful coupling between these sessions.
6:3/17

A working Humane proof-of-concept session prototype exists, using a static declarative text-free pictorial model. A composite screenshot is below. I've been showing videos of building this restaurant menu ordering session for several years at developer conferences.
6:4/17

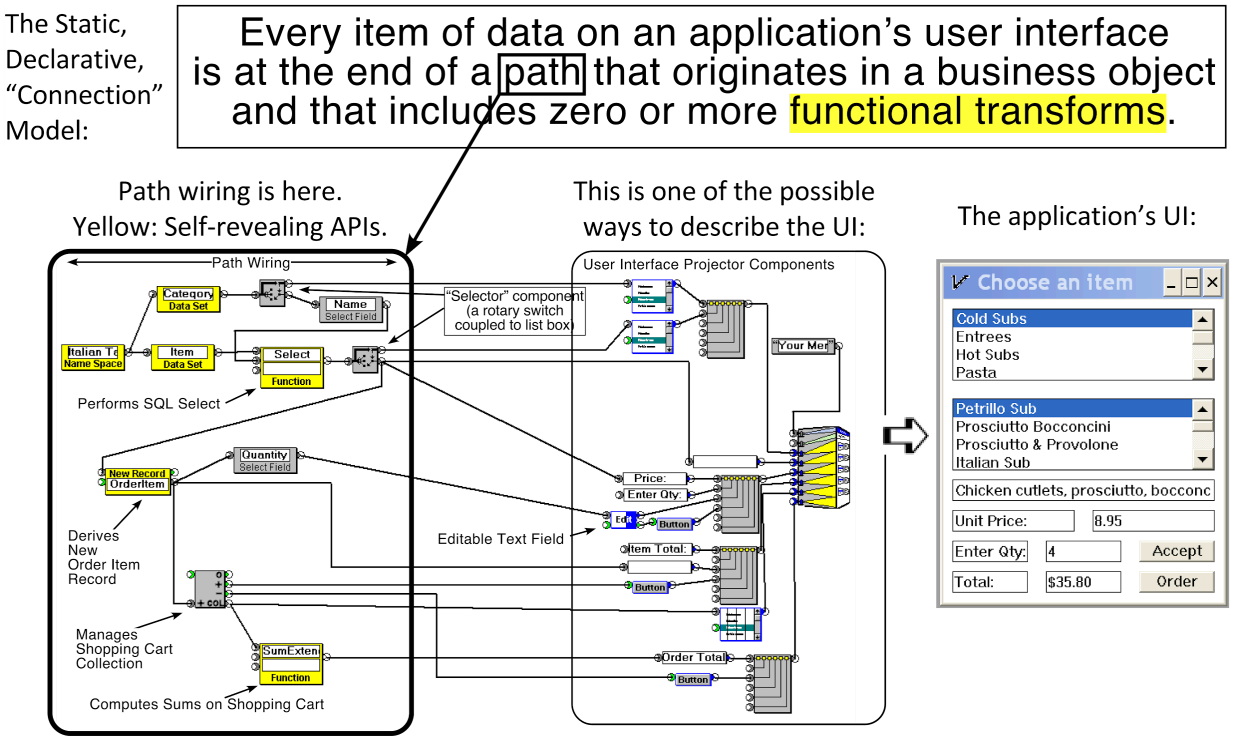
All my design documentation is public on my website. Much of it is in the form of working papers, through which I can guide interested questioners. (DMs are open.)
6:5/17

There are several models of stateful session coupling in use, e.g.,
based on a database, and
based on an immutable event log.

Their designs have not been expressed in Humane form and have not been integrated with the session model.

I believe that both are possible.
6:6/17

The Static, Declarative, "Connection" Model:



In the box at the top of the figure is a statement of the declarative connection model.

This model is a half century old. It's the basis of the user interfaces of mainframe applications, but I've not seen it suggested as an alternative to the model-view-controller pattern.
6:7/17

The wiring in the heavy rounded rectangle on the left is an expression of the model for this order-entry session. It is not a programming or flow language but a

static connection language.
6:8/17

I have demonstrated building this session in the wiring tool one component and one wire at a time, and at every stage something observable is going on.

This is the digital equivalent of the potter's wheel:

It encourages discovery and correction at every stage.
6:9/17

This discovery/correction cycle underlies the Humane Dozen concept, and it is fundamental to the goal of the proposed "for the rest of us" product:

1.Seamless evolution of the naive user from toy to tool.
2.Entry for others at any point along this path of evolution.
6:10/17

The yellow components in the left-hand rectangle are the portals to the reusable objects on the Producer side of the Platform.

The communication protocol is self-revealing APIs.
6:11/17

I anticipate that reusable objects will exist at multiple levels, including horizontal objects such as computations of all sorts, transient collections

and data stores, objects that capture domain knowledge, and application fragments built from wiring.
6:12/17

It will require experimentation, but I imagine that domain knowledge from vertical software can migrate into these reusable components and employed by systems integrators.
6:13/17

The prototype is built in Smalltalk. Smalltalk is not required for a product, but it was required for the kind of experimentation I did. I believe that a product design must be a full object design; I don't see any other way to standardize self-revealing APIs.
6:14/17

How does one do a full object design across the Web safely using existing protocols?

Is this a settled question or must it be part of the research?
6:15/17

You can see one instance of this wiring process in
http://melconway.com/talks/2018_gotober/16.html
through

http://melconway.com/talks/2018_gotober/21.html.

The rectangle in the middle of the figure shows just one way to build the UI. You could also use wireframes or forms, for example.
6:16/17

Now we have a roadmap that gives us confidence that this cloud-based Humane development/execution platform can be created.
6:17/17