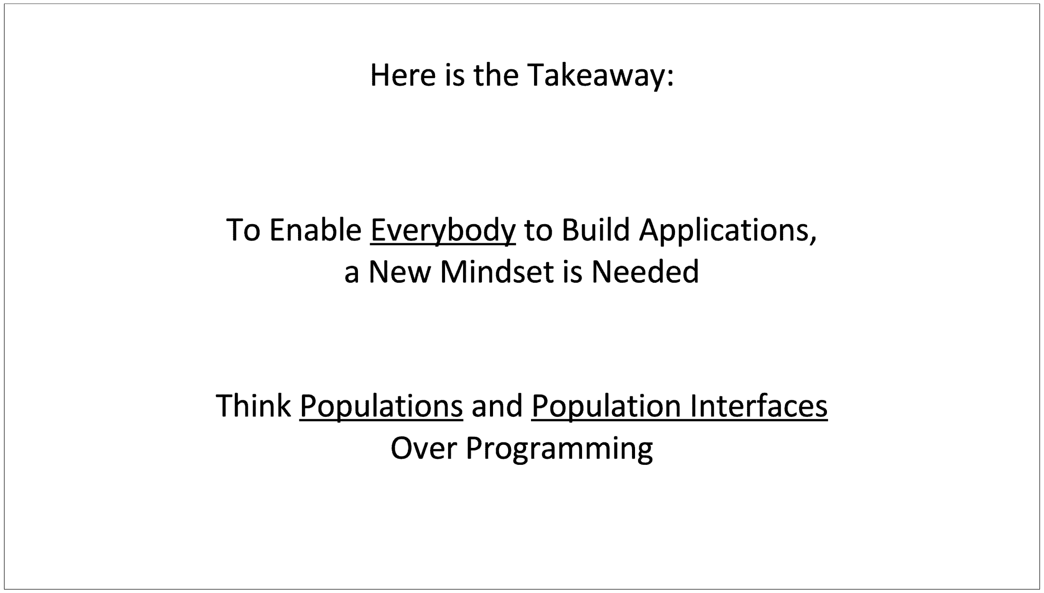


Here is the Takeaway:

To Enable Everybody to Build Applications,
a New Mindset is Needed

Think Populations and Population Interfaces
Over Programming

Let’s look at that word “Everybody”. What does it mean to take that word seriously? That’s what I’m going to talk about.

I’ve been working the simplicity problem for a quarter century. I started off seeking simpler languages. Eventually I realized that if non-programmers are going to be able to build useful real-world applications a simpler language alone is not going to do it. We have to put non-programmers in a collaborative situation where we meet them where they are and the environment empowers them by leveraging developers’ skills.

So the work has become about creating a new platform-based two-sided market in which almost everybody can participate with the skills they already have.

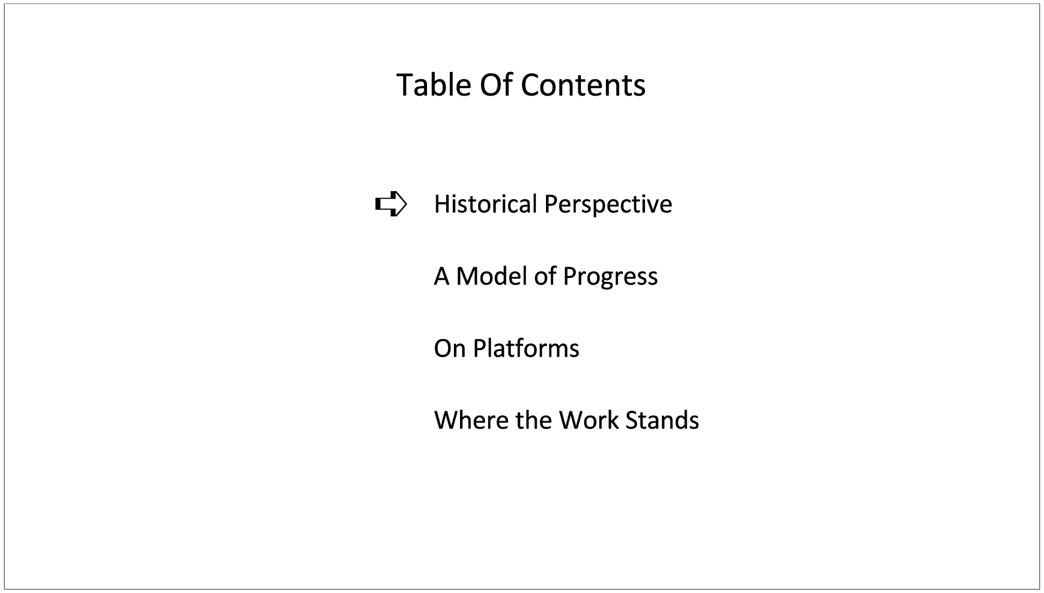


Table Of Contents

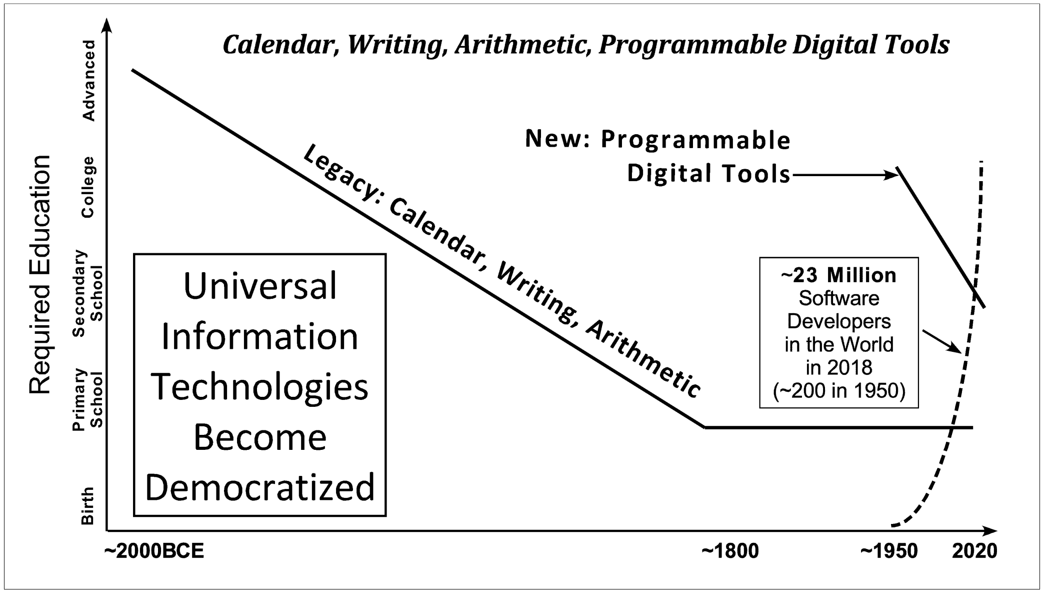
⇨ Historical Perspective

A Model of Progress

On Platforms

Where the Work Stands

Here are the four parts of my talk. It helps to look at our work in historical perspective.



We in the software industry are 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 crucial culture-disrupting information technologies**.

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.

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. I'm going to talk about how that might happen and how we might all participate in it.

The 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.

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

Also note the dotted line at the right. It shows that, over the last 70 years, the population of software developers has grown from about 200 to over 23 million. That's an average growth rate of about 18% per year over 70 years. Something big happened around 1950, which I'll discuss in a minute.

Meanwhile the educational requirement to practice digital technology has dropped from college to early secondary or middle school for a lot of children.

What happened to the legacy technologies is that their conceptual models became a lot simpler and more accessible to more people. Also, their media became more lightweight, portable, and easily reproducible.

I only had enough room to go up to 2012.
Ha! That'll freak somebody out someday.
Visit 15 Mind Features - BIZARRO.COM
© Dan Piraro · 12·21·09 BIZARROCOMIC.BLOGSPOT.COM

Here’s how it used to be. Apparently Mayan time was circular, not linear, as ours is. The Mayan calendar, which was in effect in the first millennium, was a combination of three concurrent cycles, of duration 260 days (used for ceremonial purposes), 365 days (used for agriculture), and approximately 7900 years (necessary for counting indefinitely into the past or future). You might recall that a few years ago there was a doomsday myth going around because all three cycles apparently came to their origin points in December 2012.

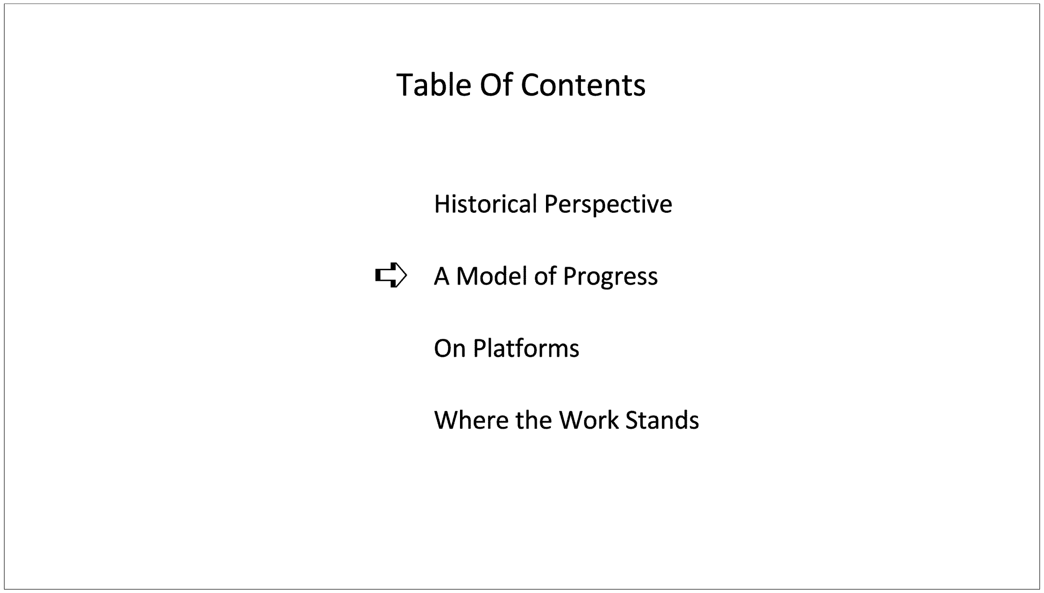


Table Of Contents

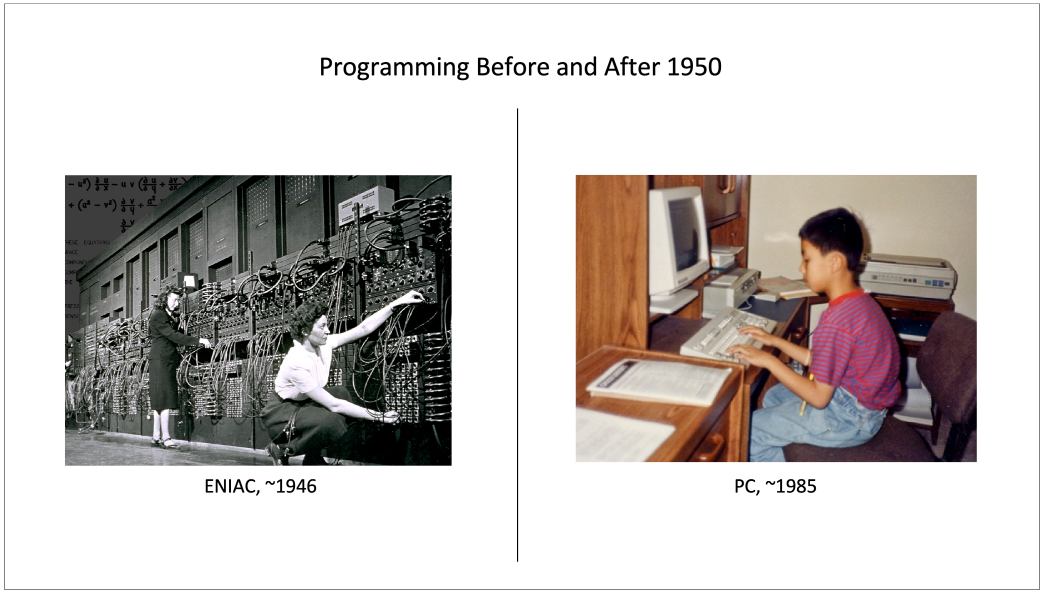
Historical Perspective

➪ A Model of Progress

On Platforms

Where the Work Stands

Let’s look at what happened over the last 75 years.



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 available. That's because a 1945 paper introduced the idea of representing a computer program as data in the computer's memory, rather than as external plugboard wiring.

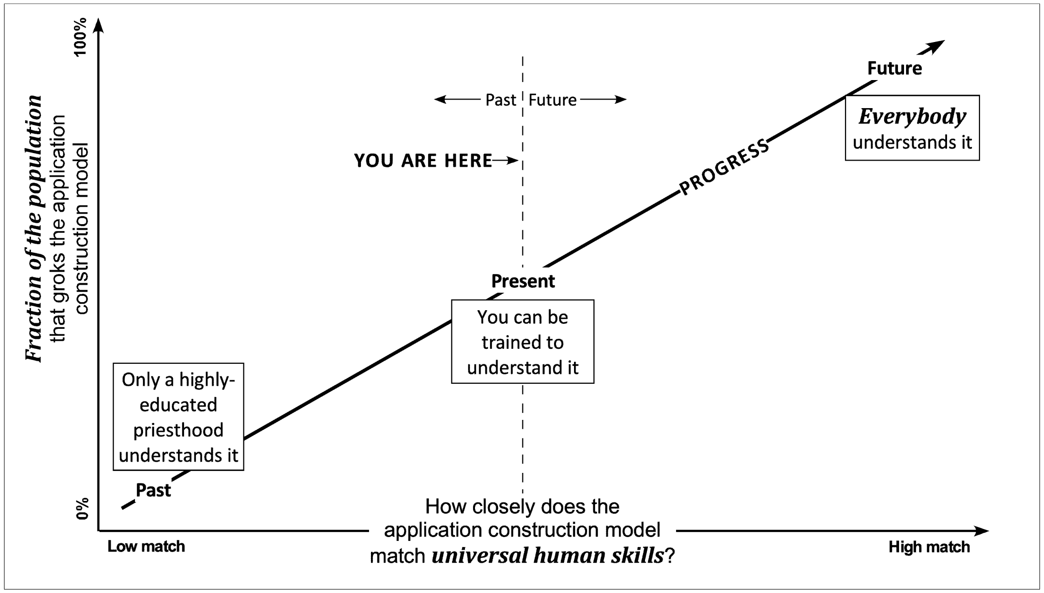
Another important change was in the weight of the embodiment of a program. (Think of the Mayan stone calendar vs the calendar on your phone.)

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 and moved on to the next application.

In addition, communication, training, and other delays were greatly reduced due to the lightweight medium, which is easily copied and transmitted.

The first commercial computers began to appear in the early 1950s. At that time, 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.

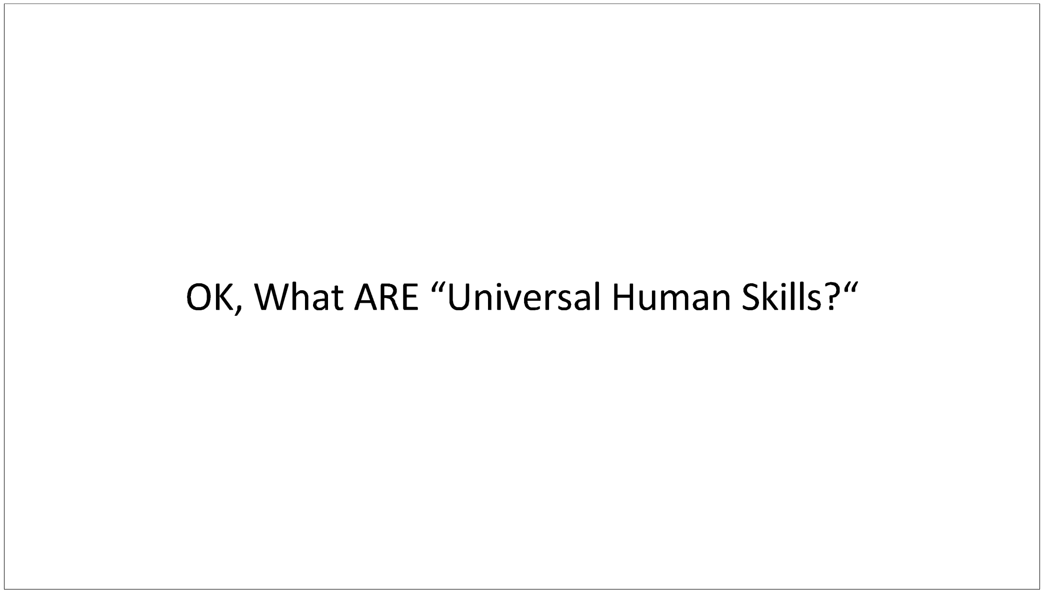
Meanwhile, Moore's law was continually driving down the costs of hardware. Personal computers started in earnest in the early 1980s, greatly enlarging the market for applications.


100%
0%
Fraction of the population that groks the application construction model
Past
Future
YOU ARE HERE
Past
Future
PROGRESS
Everybody understands it
Present
You can be trained to understand it
Only a highly-educated priesthood understands it
Past
How closely does the application construction model match universal human skills?
Low match
High match

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.

The ascending line suggests progress, from a past at the lower left where the construction 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.

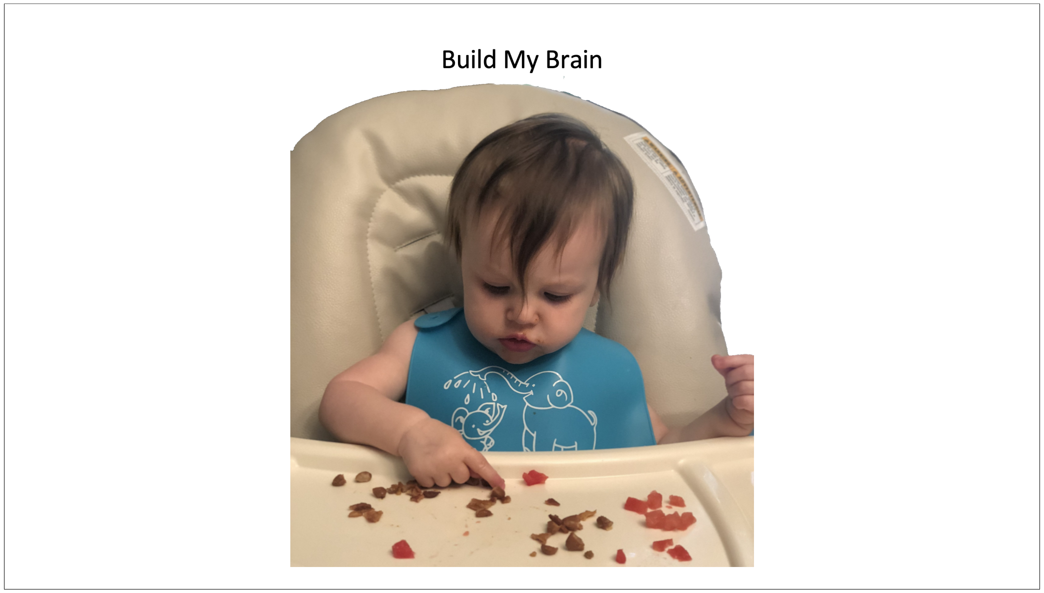
But before we go further, we have to answer what we mean specifically by "universal human skills".



OK, What ARE “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?

According to Michael Merzenich at 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.** Nature builds into every infant a process of building its brain, first passively by responding to stimuli, and then actively, through practice, relentless practice.

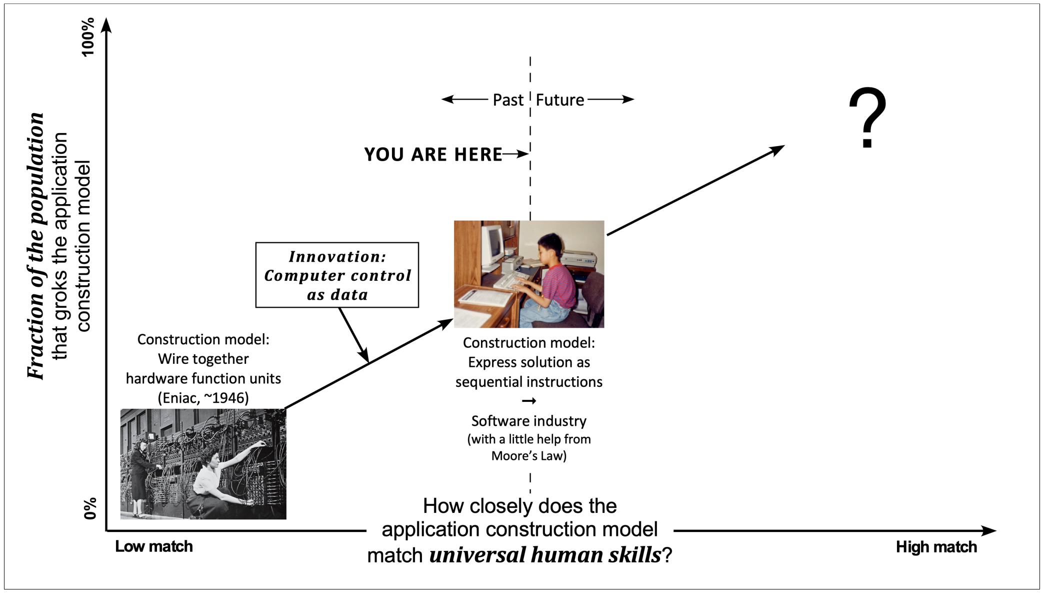


Build My Brain

Every parent sees this in toddlers, and we marvel at their amazing persistence necessary for the brain-building project, well beyond what we as adults normally can tolerate.

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

So there’s your answer. “Universal Human Skills” means hand-eye-brain coordination.


100%
0%
Fraction of the population that groks the application construction model
Past
Future
YOU ARE HERE
?
Innovation: Computer control as data
Construction model: Wire together hardware function units (Eniac, ~1946)
Construction model: Express solution as sequential instructions
→
Software industry (with a little help from Moore's Law)
Low match
How closely does the application construction model match *universal human skills*?
High match

Now back to our progress model. We'll take two data points in the past and present, and try to project into the future.

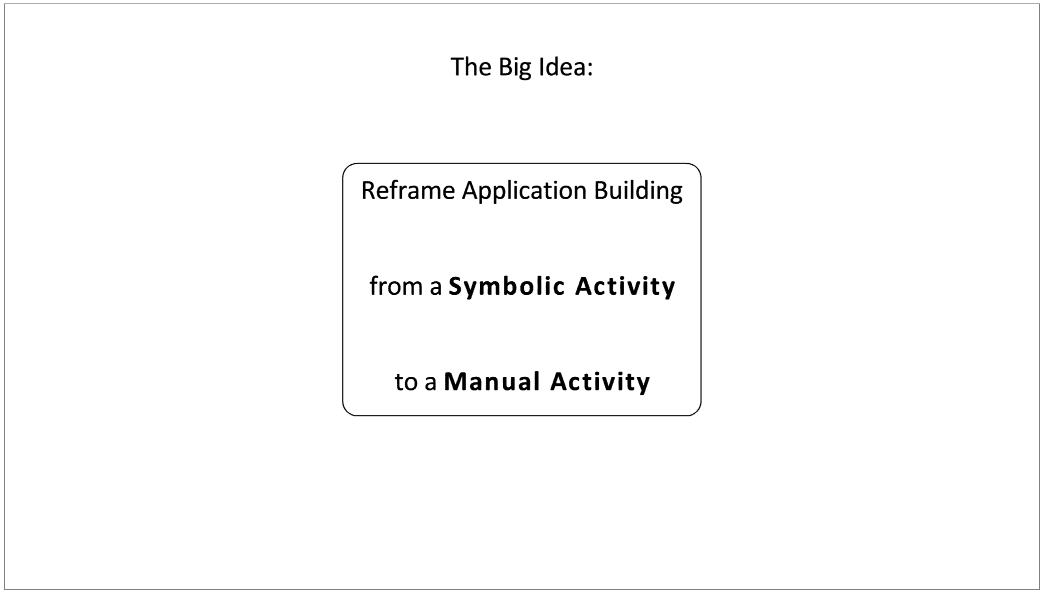
The picture at the lower left shows ENIAC programming around 1946. The ENIAC was no slouch; it was built to compute artillery tables in World War 2 and was also used to study the feasibility of thermonuclear fusion. But to program it required people trained in both mathematics and digital logic to wire up the functional units. (They were almost all women, by the way, because the men were all in the military fighting World War 2.)

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.

Another factor: the fact that programs were now data and not bundles of wires meant that computers could now create programs, and compilers and high-level languages were born, further simplifying the task.

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

That question mark, 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.



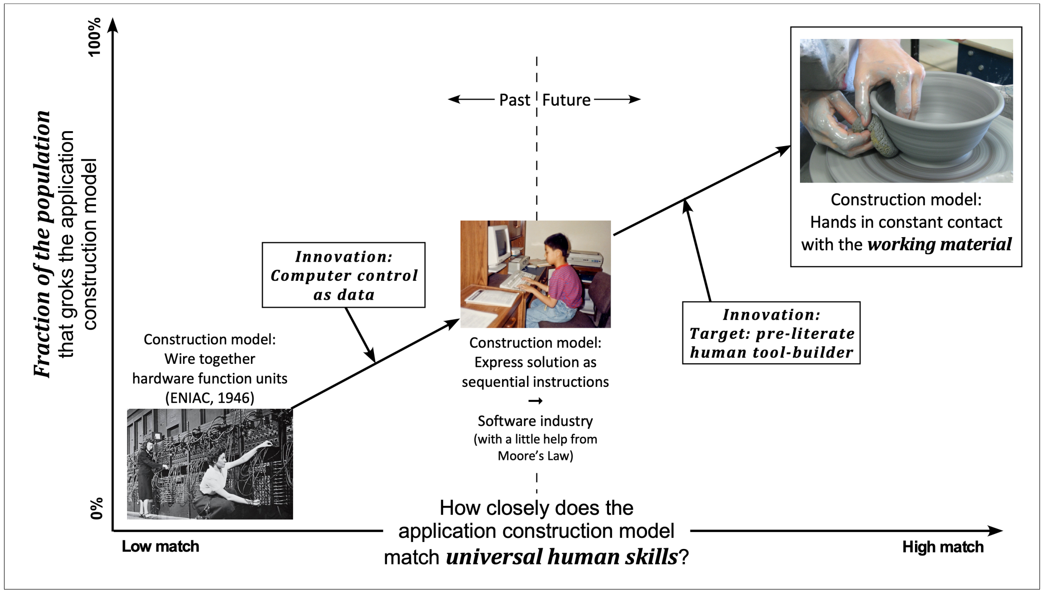
The Big Idea:

Reframe Application Building

from a **Symbolic Activity**

to a **Manual Activity**

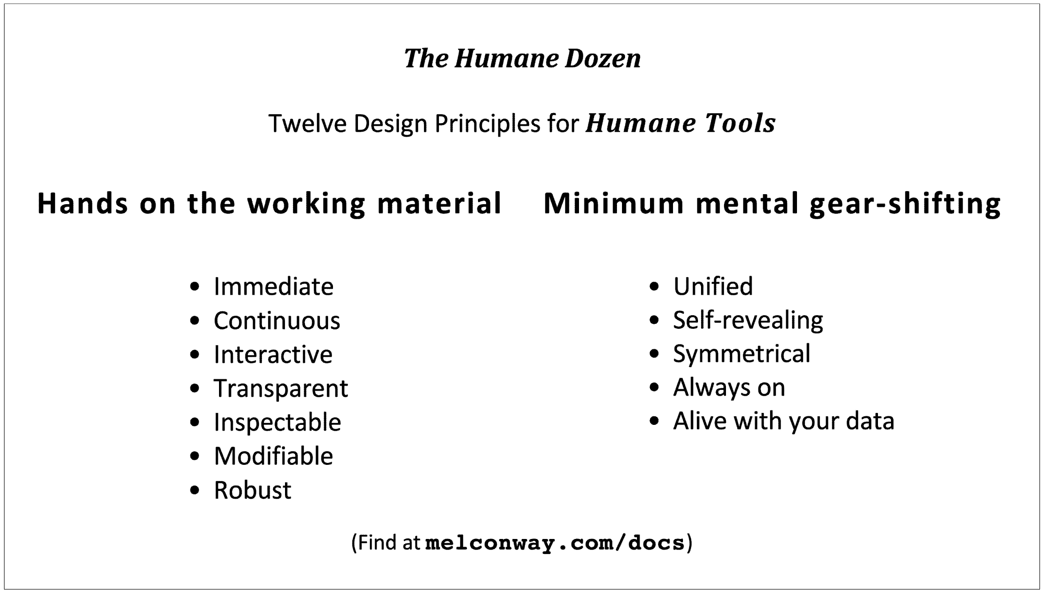
This is the big idea. We must **reframe application building from a symbolic activity to a manual activity**.
How do we do that?
And if we do it, what do we even mean by a “programming language”, or by “programming” itself?


100%
0%
Fraction of the population that groks the application construction model
Past
Future
Innovation: Computer control as data
Construction model: Wire together hardware function units (ENIAC, 1946)
Construction model: Express solution as sequential instructions
→
Software industry (with a little help from Moore's Law)
Innovation: Target: pre-literate human tool-builder
Construction model: Hands in constant contact with the working material
Low match
How closely does the application construction model match universal human skills?
High match

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**.

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. What if we need to rethink the application's conceptual model as being continually valid from the very beginning?

I've addressed these questions and have come up with a list of a dozen design principles, which I call the **Humane Dozen**.



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

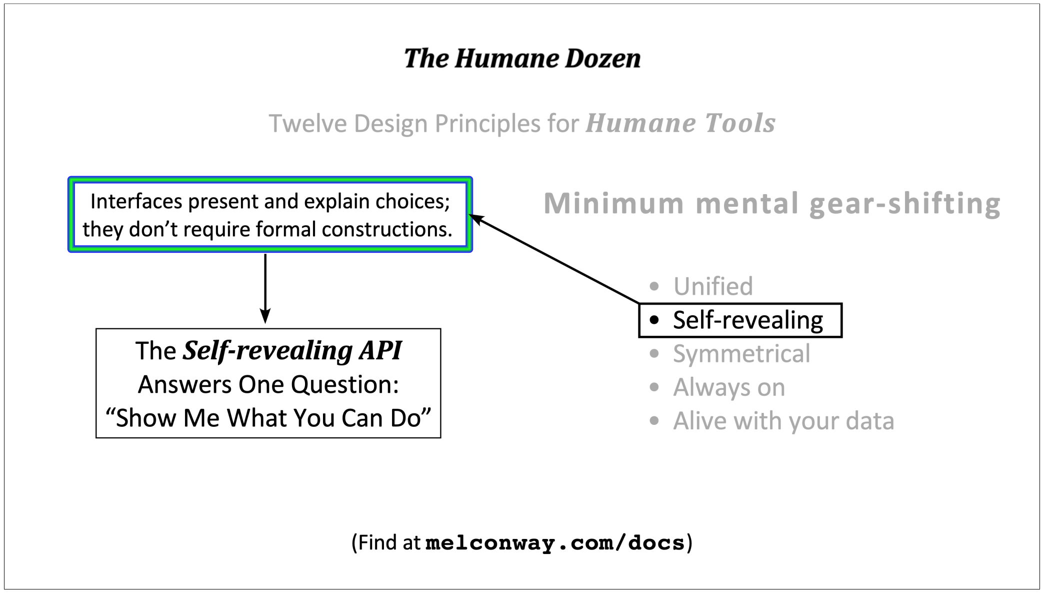
(Find at **`melconway.com/docs`**)

These twelve principles can't just be tacked onto existing development tools. They represent a holistic, Humane, approach to building and testing. The paper at the link at the bottom of the slide is only three pages and gives you one to two sentences for each principle. I've written a lot about it elsewhere but can't get into the details here.

Here is one consequence that falls out of fully following the humane dozen that you might recognize in browser debuggers. Your development tool and the application it's building (with data in it) are sitting on the display in front of you as peers. Your next click can be in either one of them. If you make a change to the application in the tool, the running application immediately reflects the change. Just like turning a bowl on a potter's wheel.

However, if you do something illegal it refuses and tells you why, and even doing stupid things won't break the application. What has to happen for that to work?

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



The Humane Dozen

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

Interfaces present and explain choices; they don't require formal constructions.

Minimum mental gear-shifting

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

The ***Self-revealing API*** Answers One Question: "Show Me What You Can Do"

(Find at **`melconway.com/docs`**)

The self-revealing principle says: **"Interfaces present and explain choices; they don't require formal constructions"**.

This, by the way, 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.

And 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.

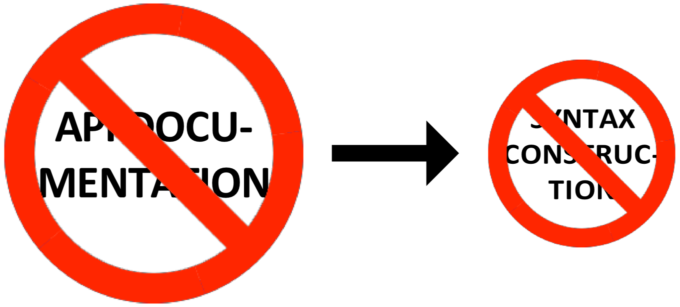
So how do you build a self-revealing API? You have to make it an active object.

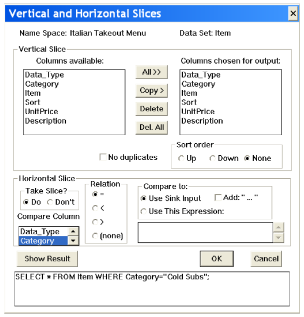
Every self-revealing API is an **active object** that can answer this one question: **Show me what you can do**. So, even with this simple example, we've changed the tool concept.

We've replaced external API documentation by tools that interact with active APIs.

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

Dialog-based Interactions Within the Development Tool





Name Space: Italian Takeout Menu   Data Set: Item

Vertical Slice

Columns available: Data_Type, Category, Item, Sort, UnitPrice, Description

Columns chosen for output: Data_Type, Category, Item, Sort, UnitPrice, Description

□ No duplicates

Sort order: ○ Up ○ Down ● None

Horizontal Slice

Take Slice? ● Do ○ Don't

Compare Column: Data_Type, Category

Relation: ● = ○ < ○ > ○ (none)

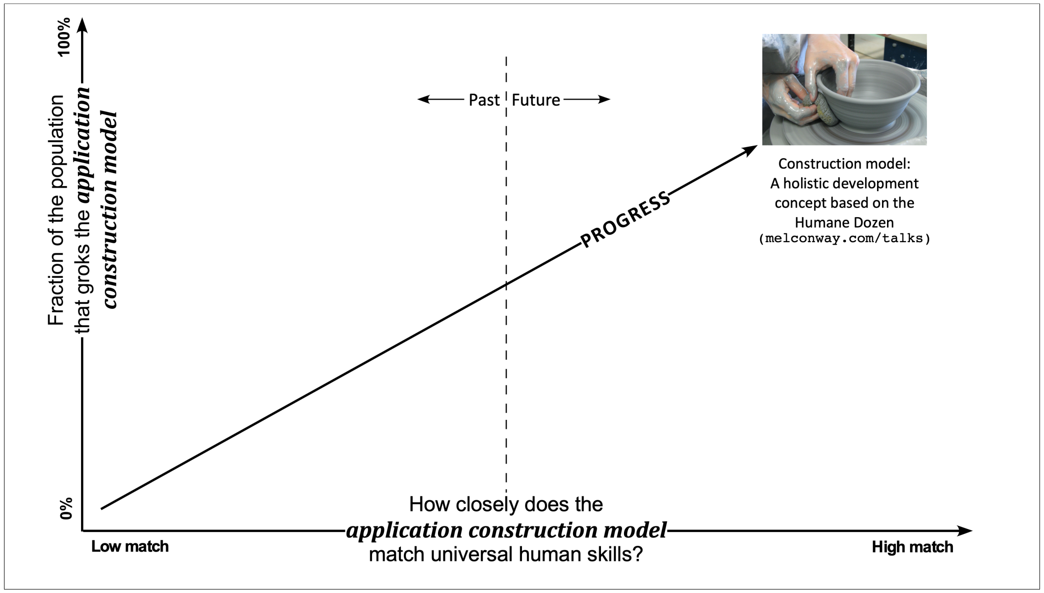
Compare to: ● Use Sink Input □ Add: " ... " ○ Use This Expression:

```
SELECT * FROM Item WHERE Category="Cold Subs";
```

Here is a dialog for building an SQL Select. It looks familiar. But again, it’s not that obvious because of another principle: Alive With Your Data. That principle means that the developer is interacting with a live program working on live data.

Here’s how this works. The tool asks the API: Show me what you can do. The API responds with a list of services that the tool lists in its dialog, including help if necessary. Once the builder chooses a service **the service takes control** and presents whatever WIMP dialogs it needs. The user might need to examine his data in order to answer some questions.

It won’t be simple, so why try? Because doing so can again greatly enlarge the number of candidates for building applications. Empowering new classes of builders means creating new markets and building new businesses.



Construction model:
A holistic development concept based on the Humane Dozen
(melconway.com/talks)

I'm not talking about today's "low-code" and "no-code" languages, whose vendors see them as productivity enhancements within the enterprise.

I'm talking about a holistic market-based construction concept that is addressed to the total population.

This work is research, but it's not pie-in-the-sky. I've worked through many of the issues. The top talk at the URL on the slide is a year old but gives you a feeling for what's possible.

So what does a software market based on this design concept look like?

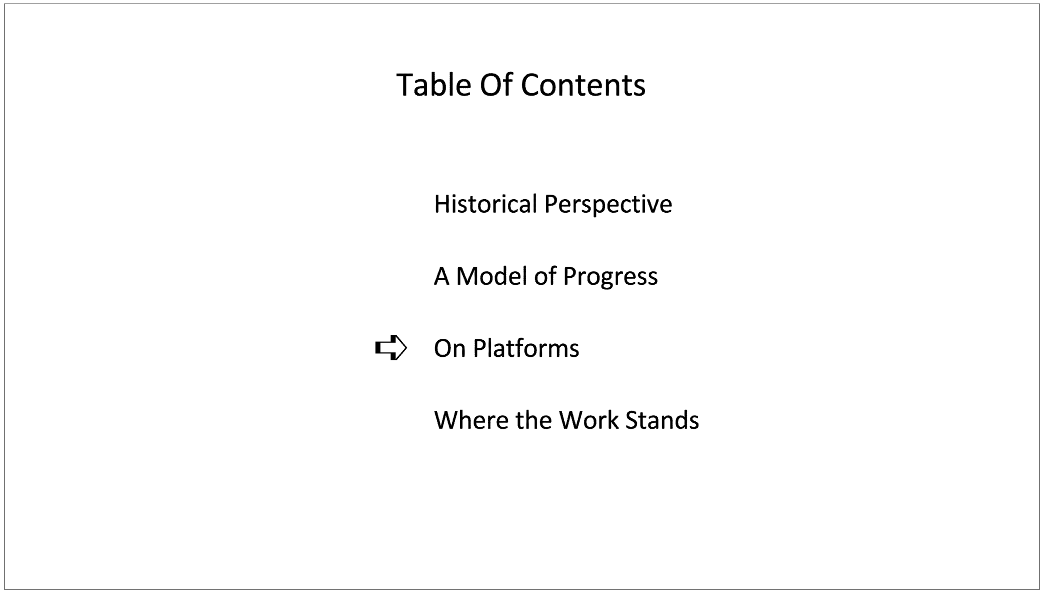


Table Of Contents

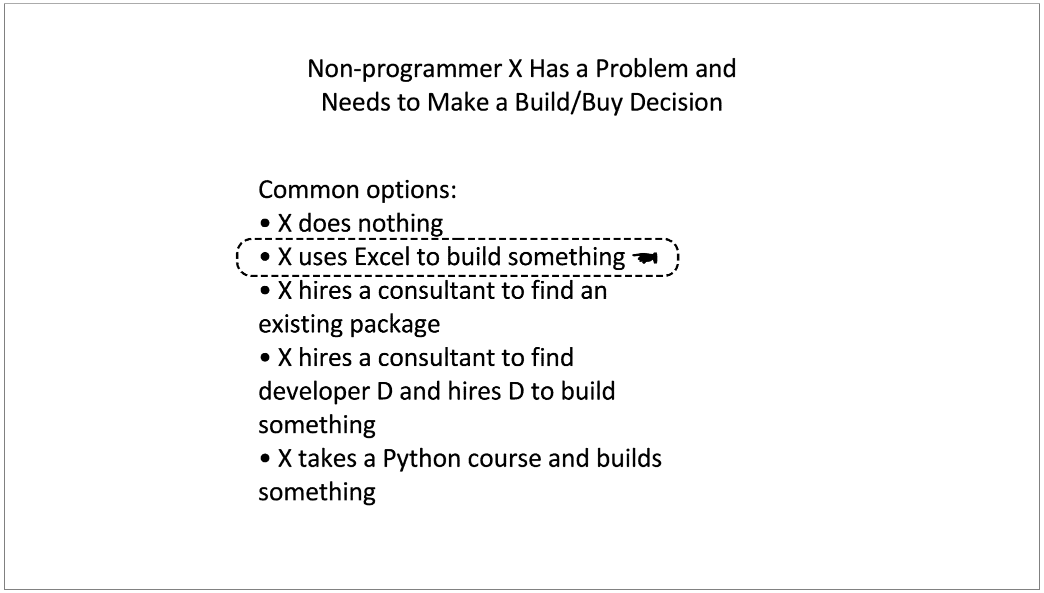
Historical Perspective

A Model of Progress

⇨ On Platforms

Where the Work Stands

What do platforms have to do with markets?



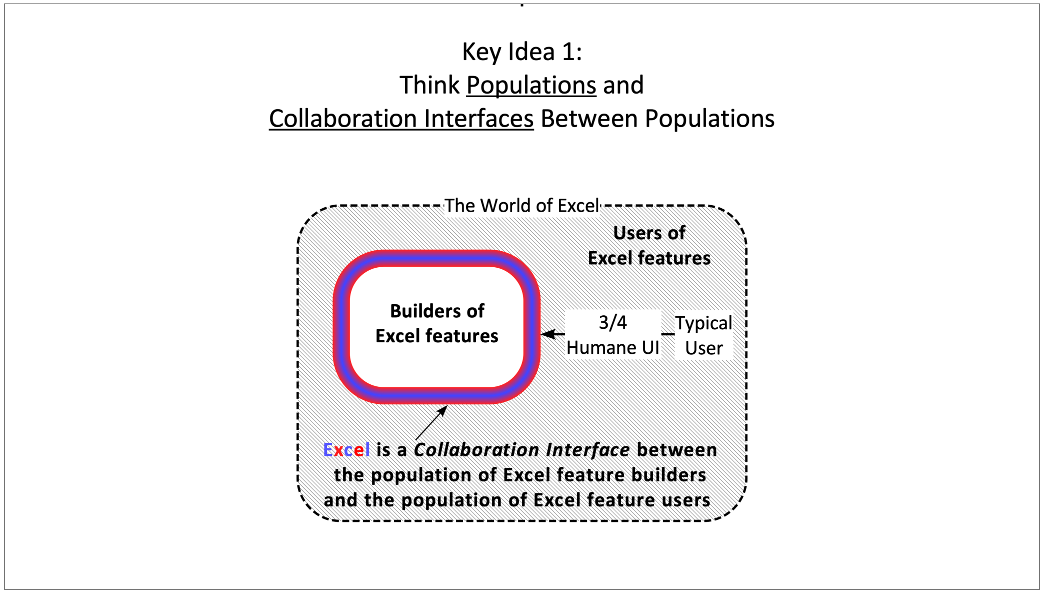
Non-programmer X Has a Problem and
Needs to Make a Build/Buy Decision

Common options:

- X does nothing
- X uses Excel to build something ☚
- X hires a consultant to find an existing package
- X hires a consultant to find developer D and hires D to build something
- X takes a Python course and builds something

Let’s start by considering hypothetical non-programmer X with a problem. X has to make a build/buy decision. It’s amazing how often Excel ends up being the choice. What’s interesting here is that Excel satisfies 3/4 of the Humane Dozen conditions. That’s one big reason people keep using it.

But Excel is much more than a novel programming language. An important 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. That’s the idea we need to carry forward.



Key Idea 1:
Think Populations and
Collaboration Interfaces Between Populations

The World of Excel

Users of Excel features

Builders of Excel features

3/4 Humane UI

Typical User

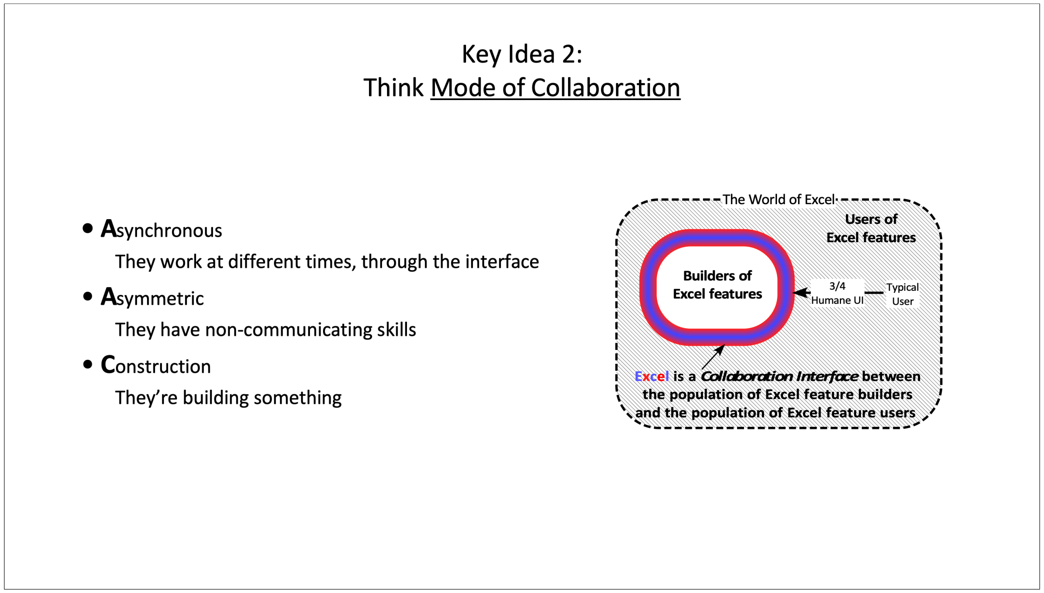
Excel is a *Collaboration Interface* between the population of Excel feature builders and the population of Excel feature users

So we're going to consider Excel, not as a piece of technology, but as an interface between two populations of people.

Everybody in the world of Excel is inside the outer rounded rectangle, and inside that there is a smaller white rectangle containing the subset of builders of Excel features.

Excel itself, the red and blue boundary, is a **collaboration interface** between these two populations.

So Key Idea 1 is the notion of two populations with different skills and a **collaboration interface** between them. We're going to end up calling it a platform.



Key Idea 2:
Think <u>Mode of Collaboration</u>

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

The World of Excel
Users of Excel features
Builders of Excel features
3/4 Humane UI
Typical User

Excel is a *Collaboration Interface* between the population of Excel feature builders and the population of Excel feature users

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 the work of the two populations into something new.

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.

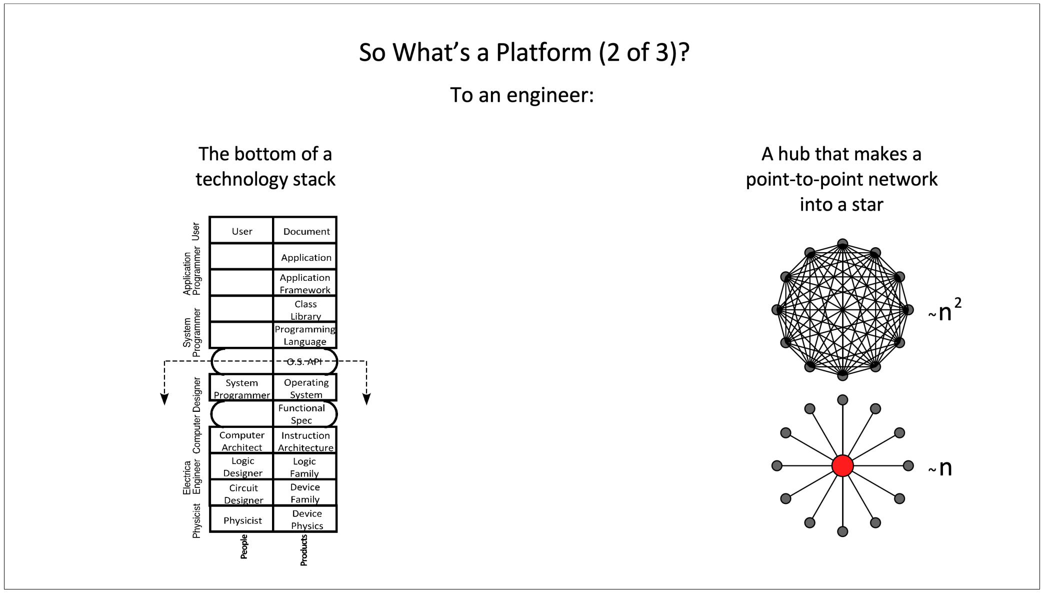
Construction. Some platforms, railroads, for example, don't build new things as the result of their use. Some, like the potter's wheel, do.

So What’s a Platform (1 of 3)?

To a layperson: Something you put stuff on

Photo by the Cowan Lab, Louisiana State University

So 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.



So What’s a Platform (2 of 3)?

To an engineer:

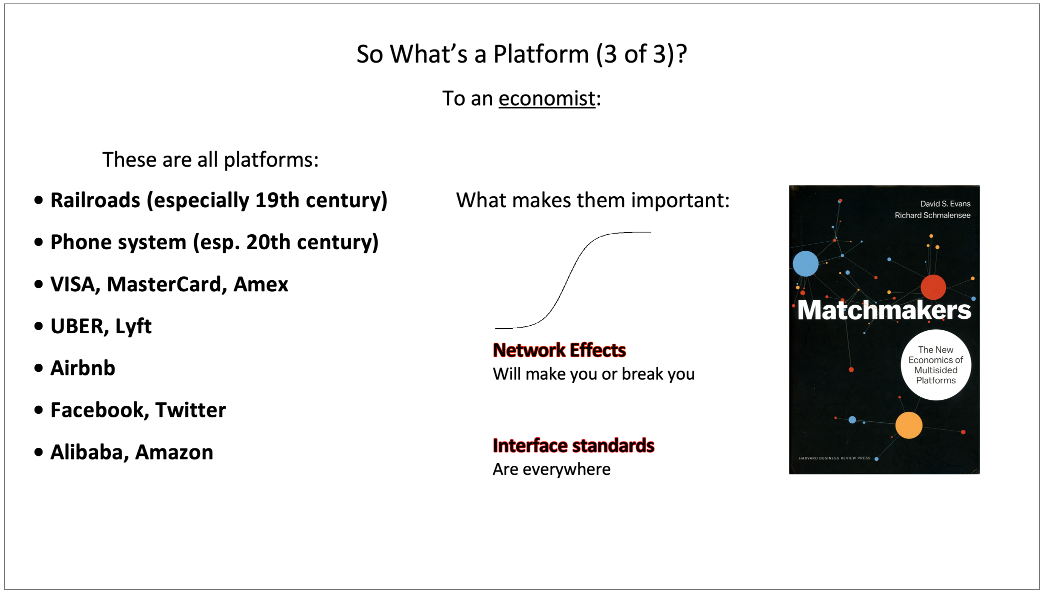
The bottom of a
technology stack

People	Products
User	Document
	Application
	Application Framework
	Class Library
	Programming Language
	O.S. API
System Programmer	Operating System
	Functional Spec
Computer Architect	Instruction Architecture
Logic Designer	Logic Family
Circuit Designer	Device Family
Physicist	Device Physics

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. That is part of the Mulesoft value proposition as I understand it.



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

David S. Evans
Richard Schmalensee
Matchmakers
The New Economics of Multisided Platforms
HARVARD BUSINESS REVIEW PRESS

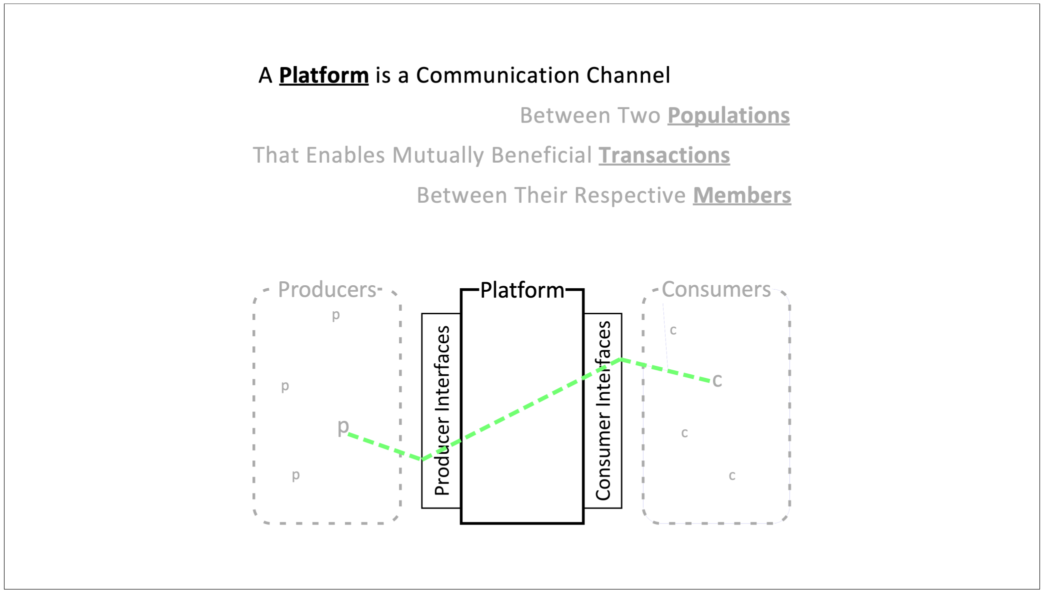
To an economist, though, a platform is a powerful economic phenomenon that connects two populations in what's called a two-sided market. You'll recognize the examples on the left; they're all associated with rapid market growth.

The two populations are sometimes called producers and consumers, but asymmetry is not an essential part of the definition.

Network effects are really important to market growth. The more participants on one side of the platform, the more valuable the platform is to the participants on the other side. Consider the early days of the telephone. You want a phone service that connects you to the people you want to talk to. 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; Facebook is a good example.

When you examine successful platforms you'll see a lot of **standardization of interfaces**, which further enhances network effects.

if you want to study platforms as economic phenomena I recommend the book "Matchmakers" by Evans and Schmalensee.



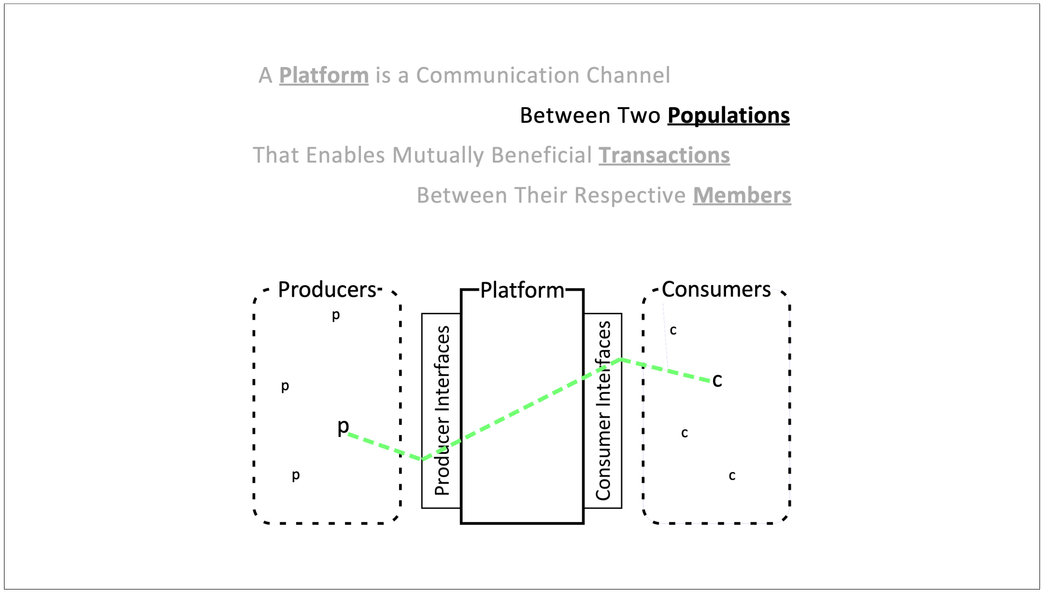
A **Platform** is a Communication Channel

Between Two **Populations**

That Enables Mutually Beneficial **Transactions**

Between Their Respective **Members**

(rapid sequence of four slides)
Here’s my formal definition of a platform:
A Platform is a communication channel…



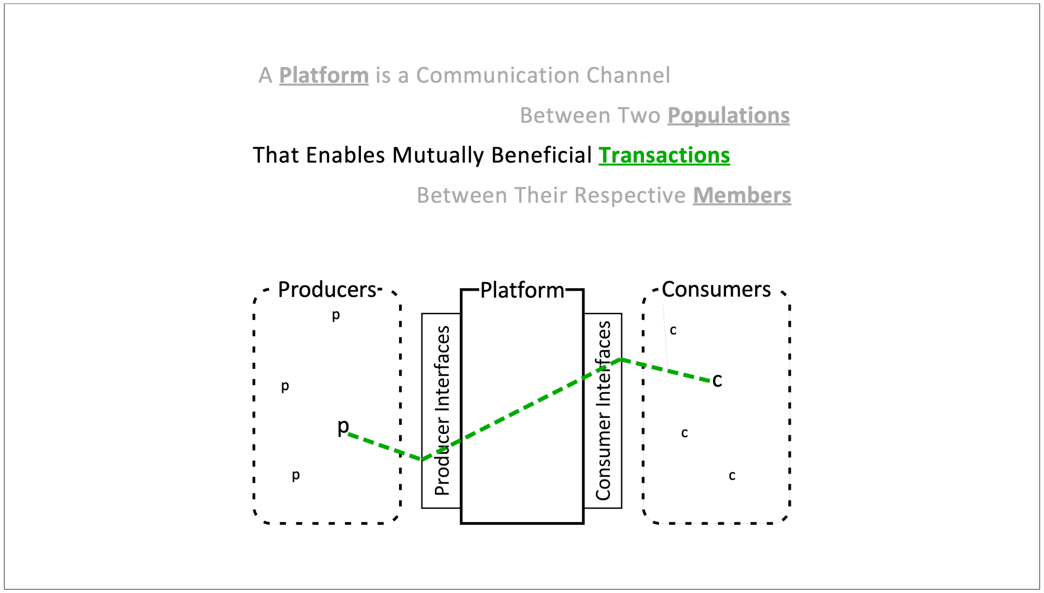
A **Platform** is a Communication Channel

Between Two **Populations**

That Enables Mutually Beneficial **Transactions**

Between Their Respective **Members**

…between two populations (producers and consumers)…



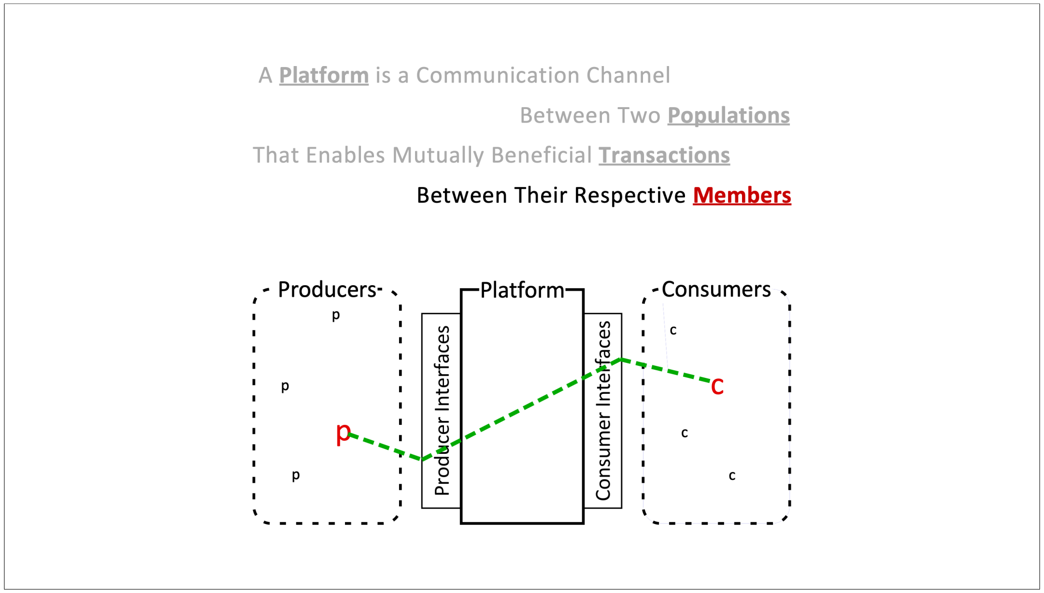
A **Platform** is a Communication Channel

Between Two **Populations**

That Enables Mutually Beneficial **Transactions**

Between Their Respective **Members**

…that enables mutually beneficial transactions…



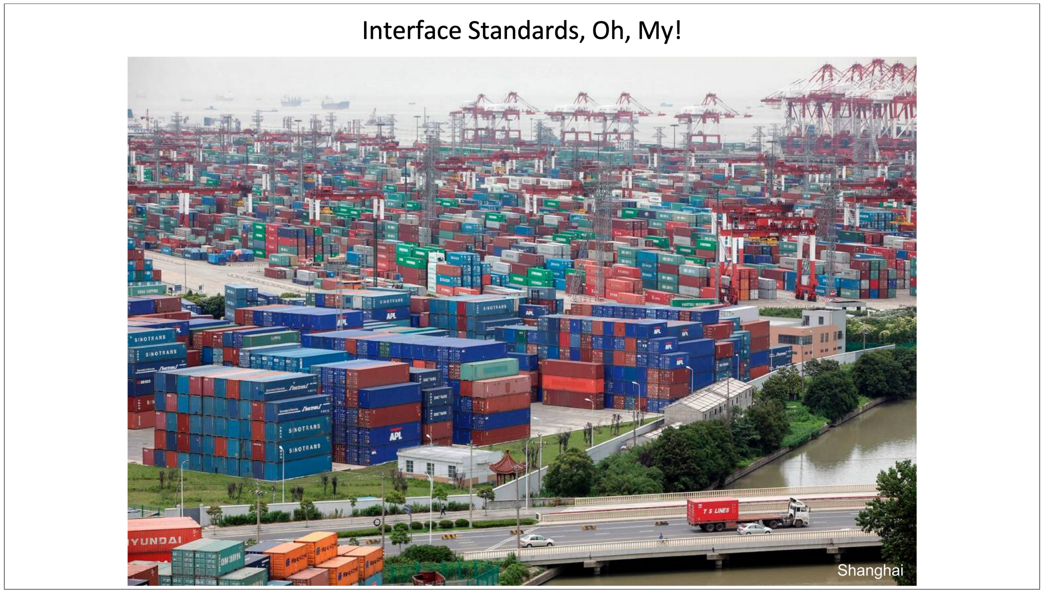
A **Platform** is a Communication Channel

Between Two **Populations**

That Enables Mutually Beneficial **Transactions**

Between Their Respective **Members**

…between their respective members.



Interface Standards, Oh, My!

Shanghai

I’ve said that platforms involve interface standards. Here’s my favorite interface standards picture, the Shanghai container port. From the ships and cranes in the background to the containers and trucks in the foreground, to the barcodes on every container connecting it to an unseen worldwide information system, I dare anyone to count all the interface standards.

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. 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.

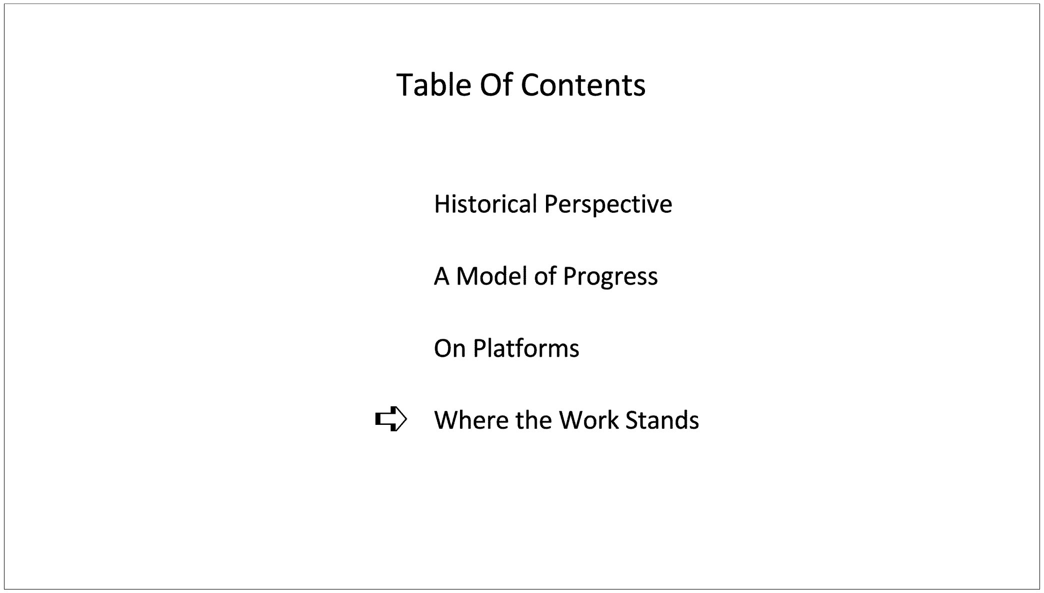


Table Of Contents

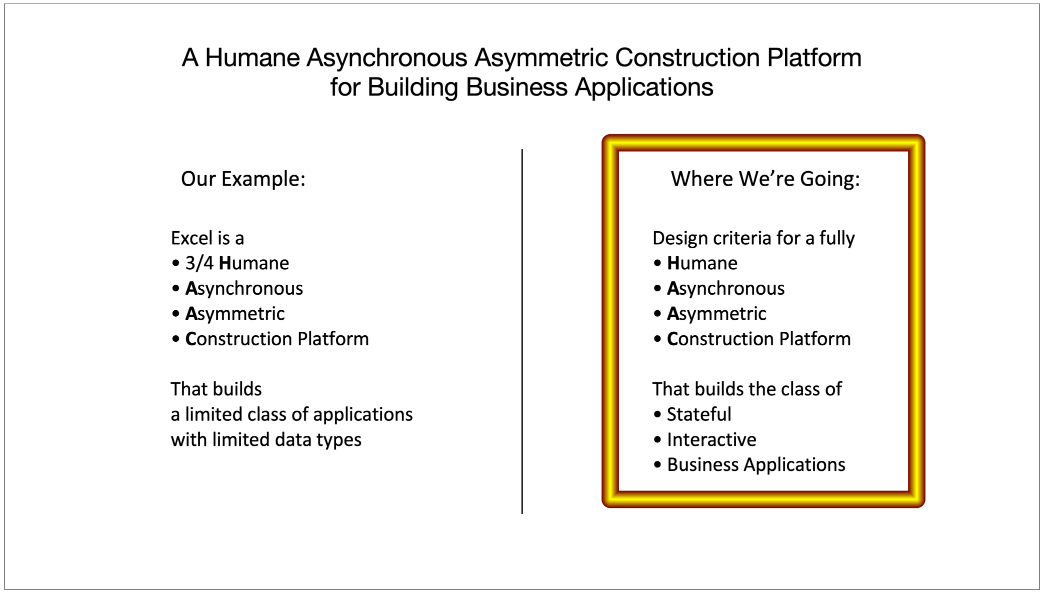
Historical Perspective

A Model of Progress

On Platforms

⇨ Where the Work Stands

OK, we’ve got the building blocks in place to picture where we are and where we’re going.



A Humane Asynchronous Asymmetric Construction Platform
for Building Business Applications

Our Example:

Excel is a

- 3/4 **H**umane
- **A**synchronous
- **A**symmetric
- **C**onstruction Platform

That builds
a limited class of applications
with limited data types

Where We're Going:

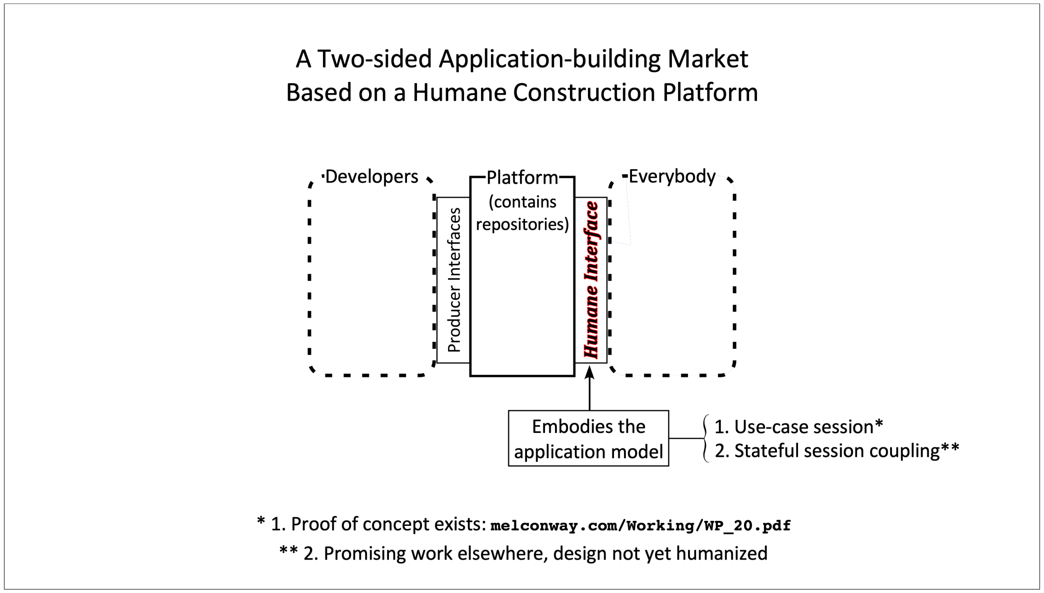
Design criteria for a fully

- **H**umane
- **A**synchronous
- **A**symmetric
- **C**onstruction Platform

That builds the class of

- Stateful
- Interactive
- Business Applications

Most of the pieces are in place to build a **Humane Asynchronous Asymmetric Construction Platform for business applications**.



A Two-sided Application-building Market
Based on a Humane Construction Platform

1. Use-case session*
2. Stateful session coupling**

* 1. Proof of concept exists: `melconway.com/Working/WP_20.pdf`

** 2. Promising work elsewhere, design not yet humanized

This is what the platform looks like. On the left are developers building reusable components and on the right is just about anybody assembling these components into applications. The Consumer Interface on the right presents the application model to Everybody in Humane form.

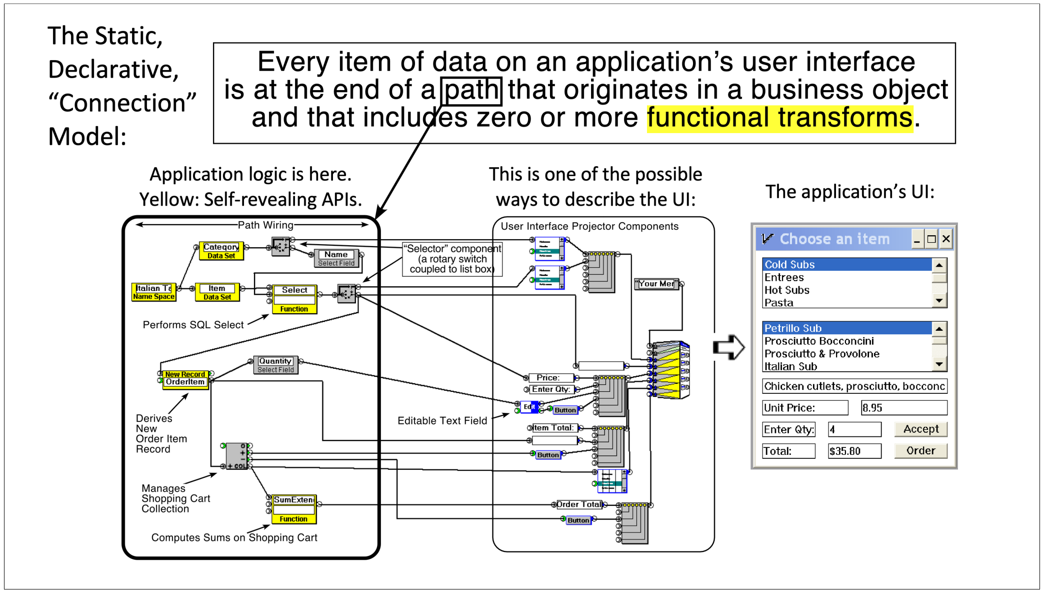
Yes, but what does it really look like?

I can answer by dividing the application model into two parts: each use-case session, and stateful coupling between these session.

1. **The model of a use-case session**. I have built a proof of concept of a fully Humane tool that builds one-session applications using a static declarative pictorial model without text. The working paper in note 1 describes the model with a detailed walkthrough with links to demonstration videos in which I build a small demo app one piece at a time. I've been showing videos of this for a couple of years at developer conferences.

2. **The model of stateful session coupling**. Others are developing this model (see, e.g., https://eventmodeling.org/) but its design has not been expressed in Humane form and has not been integrated with the session model. I believe that these are both possible.

To finish, I'll show you just one slide with some screen shots from that working paper.



The Static, Declarative, “Connection” Model:

Every item of data on an application’s user interface is at the end of a path that originates in a business object and that includes zero or more functional transforms.

Application logic is here. Yellow: Self-revealing APIs.

This is one of the possible ways to describe the UI:

The application’s UI:

Choose an item

Cold Subs
Entrees
Hot Subs
Pasta

Petrillo Sub
Prosciutto Bocconcini
Prosciutto & Provolone
Italian Sub

Chicken cutlets, prosciutto, bocconc

Unit Price: 8.95

Enter Qty: 4

Total: $35.80

At the top is a statement of the declarative connection model. (read)

Every item of data on an application’s user interface is at the end of a path that originates in a business object and that includes zero or more functional transforms.

Its realization is wiring, as you see, but keep in mind that this this wiring not a programming or flow language but a **static connection language**.

I build this app from scratch one component and one wire at a time and **at every stage something observable is going on.**

On the bottom left is the wiring of the application. The rectangle in the middle shows just one way to build the UI. You could also use wireframes or forms, for example.

So now we have a roadmap with enough guidance that gives us confidence that this Humane Platform can be created.

Thank you!

@conways_law

conway.mel@gmail.com

Thanks for your attention. Feel free to contact me on Twitter, where Direct Messages are open, or by email.