Building Applications Becomes Simpler and More Accessible In a Two-sided Market

Preface on simplification: When it comes to building software, the meaning of "simplification" depends on context. Please consider here how my thinking about simplification has evolved.

I started with the assumption that there is something more to learn about simplification. There is historical evidence for this assumption. Arithmetic, calendar, and writing started off centuries ago as the properties of priest classes, and now they are taught in primary school. The ability to build ***and understand the workings of*** software applications is still the property of a priest class; that tells us how far we have to go.

I have assumed that the simpler something is, the more people there will be who understand it. Until a few years ago I defined "simple" to mean "it should join arithmetic, calendar, and writing as part of the primary-school education". Having learned that the route to a child's mind is through the hands, I concluded that the model of building software ***that best matches the way we are all built*** is not a symbolic but a manual model. I see a potter at the wheel rather than a geek at the keyboard. This has led to a set of humane design principles for the application-building process that have governed my work.[1] Given the premise of radical simplification, this in turn led to a rejection of text languages. What I have come to realize after looking back on my discovery of the "editor hypothesis"[2] in the 1980s is that using terms like "programming" and "programming language" imposes a seriously limiting bias on thinking about simplifying building business applications.

Then, my exposure to Domain-Driven Design changed the subject from teaching children to building business applications, where there are ***two*** important populations of interest: developers who understand the technology, and business people who understand the need but are excluded from important stages of the building process. There is a consensus among many observers that this exclusion of business people from all but the earliest stages of application building has exacted a high price in terms of failure to build correct and timely solutions.

I realize now that seeking to make things so simple that non-programmers can build business applications by themselves, or that all parts of the application must be simplified, is misguided. ***If there is an approach to advancing universal accessibility it lies in collaboration***. This has led to a two-part application model; one part has to be built by developers but the other part can be built by non-programmers. I now have a candidate for such a model[3] that I believe is worth taking to the next stage. Furthermore, ***I believe this construction model can be implemented as a web-based two-sided market that benefits from network effects.***

---

1 http://melconway.com/Home/pdf/humanedozen.pdf

2 I describe the editor hypothesis in the paper.

3 The most recent demo (consider it to be a placeholder) is at http://melconway.com/talks/2018_gotober/

Twitter: @conways_law

Date of pdf: 2019-01-02
conway.mel@gmail.com

Clarification: The word "platform" has acquired two different meanings in the software community.

1. The older meaning: A *platform* is a technical setup in which software or other technology is executed or built, for example, a "software development platform" or a "Platform as a Service".[4]
2. A more recent meaning, coming up fast since the advent of social networks: A *platform* is the intermediary in a two-sided market:[5] an economic environment in which two distinct groups of participants (sometimes called *producers* and *consumers*) exchange value via the intermediary. A distinctive characteristic of a two-sided market is that it benefits from network effects;[6] that is, adding consumers makes participation more valuable to producers, and vice versa. Examples of platforms range from weekly village farmer's markets to social networks.

In this paper the definition we use is the second, two-sided-market, definition.[7]

Foreword.

Multiple "low-code" and "no-code" offerings that simplify enterprise application development are coming on strong.[8,9] Most address professional developers[10,11]; a few primarily address what are being called "citizen developers"[12], technical contributors who are not necessarily programmers but who help build applications within the enterprise setting[13].

By contrast, what I'll be describing here is envisioned as a sub-enterprise, open-market-based approach, in which the participants don't have to know each other or work in the same organization. It is motivated primarily by the goal of universal accessibility. The introduction strategy proposed here is bottom-up, minimum-viable-product[14].

Here is the reasoning on which this paper is based. The goal is to give an important part of the application-building process to non-programmers who want to build something useful in a way that (1)is simple enough to be accessible to almost everybody, and (2)gives them the part of the application that they know most about, the use-case-intensive part (as distinguished from business-rule- or infrastructure- intensive parts). This is done collaboratively; programmers and non-programmers build their different parts separately within a Web-based platform that enables the synthesis of both. Furthermore, this is done in a way that engages programmers and non-programmers in

---

4 https://en.wikipedia.org/wiki/Platform_as_a_service

5 https://en.wikipedia.org/wiki/Two-sided_market

6 https://en.wikipedia.org/wiki/Network_effect

7 An excellent reference: G. Parker, M. Van Alstyne, S. Choudary (2016), *Platform Revolution: How Networked Markets Are Transforming the Economy - and How to Make Them Work for You*. New York: W. W. Norton

8 Gartner: http://melconway.com/Working/WP_17.pdf

9 Forrester: https://reprints.forrester.com/#/assets/2/225/'RES137262'/reports

10 Salesforce: https://www.salesforce.com/

11 Mendix: https://www.mendix.com/

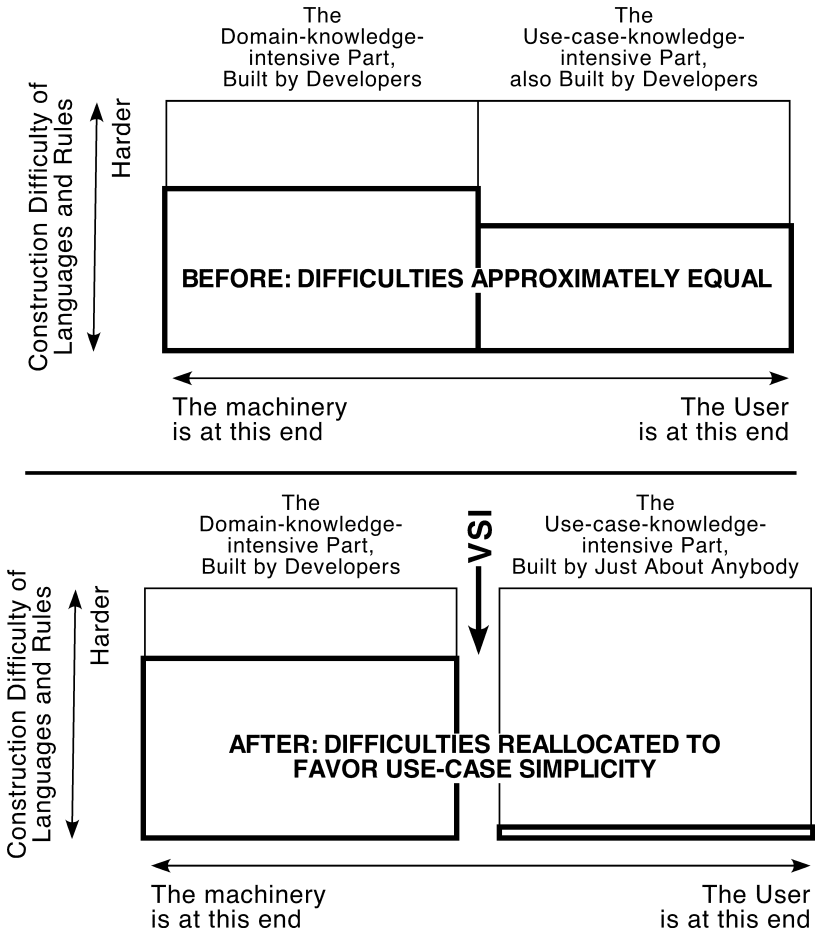
12 https://www.gartner.com/it-glossary/citizen-developer/

13 Betty Blocks: https://www.bettyblocks.com/

14 https://en.wikipedia.org/wiki/Minimum_viable_product

a two-sided market such that, given good management of the platform, network effects incentivize both to participate.

As the paragraph above suggests, the approach is to repartition the work of building a whole application into two barely coupled parts: a domain-knowledge-intensive part built by software developers, and a use-case-knowledge-intensive part built by non-programmers. The effect of this repartitioning is depicted in the following figure.



I have demonstrated the technical feasibility of this repartitioning[15]. The key element is the Visual Service Interface (VSI) between the two parts. How ***the VSI transforms the use-case builder's experience of an API*** in the domain-knowledge part is analogous to how ***Windows and Macintosh transformed the computer user's experience of the command line***: it presents code as dialog windows presenting choices, greatly simplifying the experience and making it accessible to many more people.

15 See, for example, http://melconway.com/talks/2018_gotober/

After the repartitioning, the people building the use-case-knowledge-intensive part have a much simpler job; developers have a little more to do, but I expect this to be compensated by increased reusability of their work.

The major technical pieces are now in place to build a platform-based two-sided market that will enable these two classes of participants to collaborate in building interactive business applications. The "platform" is a drag-and-drop wiring tool whose "producers" are software developers who build software "domain objects" accessed by some of the wired-up components, and whose "consumers" do the wiring.

The argument of the paper, that there is potential for the creation of new economic activity, is built on two ideas:

1. Inventions that enable two communities to collaboratively build something (where they previously couldn't because of incompatible disciplines) can lead to new economic activity. The beginning of the paper reviews the history of the microprocessor as a model of this process. It then finds three enabling conditions of these inventions, which it then applies to the present invention.
2. The present invention is a repartitioning of business applications as described above, together with an interface (the VSI) between the two paritions that enables collaborative building between developers and non-programmers who work within the platform.

The structure of the invention is captured in Figure 5 (page 15), and the three enablers in the invention are listed on pages 16 and 17.

Table of Contents

Twitter: @conways_law

Date of pdf: 2019-01-02
conway.mel@gmail.com

History

Here is some personal technology history. I'll be generalizing from this later.

Personal Recollections

In the early 1970s I was building digital-to-video medical image converters of my own design out of Transistor-Transistor Logic (TTL)[1]. That was when Intel was introducing the first microprocessor chipset, called the MCS-4, based on the 4004 central processing unit[2]. I needed to build a magnetic-tape controller for my CAT-scanner viewers and I tried to figure out how to use the MCS-4, but its tooling wasn't intended for a lone inventor working out of a bedroom, so I built a TTL state machine instead. That way the only additional tooling I needed was a small ROM programmer, with which I could write the program of my state machine into programmable ROMs, four toggle switches at a time.

There were other single-chip processors around then. They were typically used as embedded equipment microcontrollers. With the MCS-4, however, Intel made a decision to build a chipset with the architecture of a programmable computer to meet a requirement (for a calculator) that could have been satisfied by a more specialized device. This decision to favor a device whose function was defined, not in an Intel factory but by software that could be changed by the customer after manufacturing, was validated as other Intel customers found uses for this new microprocessor.[3]

Those few years in the early 1970s began a revolution in how control of manufactured equipment is implemented. Since then equipment has been continually getting smarter and smarter, using more and more software.[4]

It took a decade before Intel's decision led to the IBM PC, which was based on a distant successor of the MCS-4. Concurrently, MOS Technology was building the brilliantly simple 6502[5], the basis of the Apple II. Later, Motorola built the 68000[6], the basis of the original Macintosh.

The PC prevailed largely because of network effects:

1. The Apple II was a hobby until the first spreadsheet, VisiCalc[7], was built for it.

2. The utility in business of the Apple II/VisiCalc bundle became obvious quickly, and people started buying them for their offices, something they could expense without corporate IT approval because the price was below the approval threshold.
3. IBM and its corporate IT clients saw this incursion of the Apple II as a threatening bureaucratic workaround, which caused the first IBM PC to be rushed out.
4. VisiCalc was ported to the PC but was quickly superseded on the PC by Lotus 1-2-3, a superior product that included database and graphing features.
5. IBM's entrenchment in corporations, the number of PCs (and compatibles), and the ability to write applications across them all due to MS-DOS and PC-DOS compatibility drove a positive feedback loop in application development/computer demand that led to the dominance of the PC architecture.
6. Microsoft then finessed IBM out of the OS market[8] and carefully migrated its DOS users to Windows. It's estimated that Windows currently has about 83% of the desktop OS market.[9]

VisiCalc transformed the Apple II from a hobby toy to a business machine; it demonstrated the importance of the "killer app" as a nucleus in seeding market creation. The effect that the Apple II/VisiCalc bundle had on industrial IT is a powerful illustration of Clayton Christensen's model of disruption.[10,11]

Twitter: @conways_law
Date of pdf: 2019-01-02
conway.mel@gmail.com

The "Two-sided Market"

The resulting PC desktop application-software market is an instance of a two-sided market[12,13].

> Paraphrasing Wikipedia: "A two-sided market", also called a two-sided network, is a set of economic activities having two distinct user groups, mediated by an intermediary, that provide each other with network benefits. The intermediary that enables interactions between the users in the two groups is called a "platform".

We need a diagram with more specificity. Here is one:

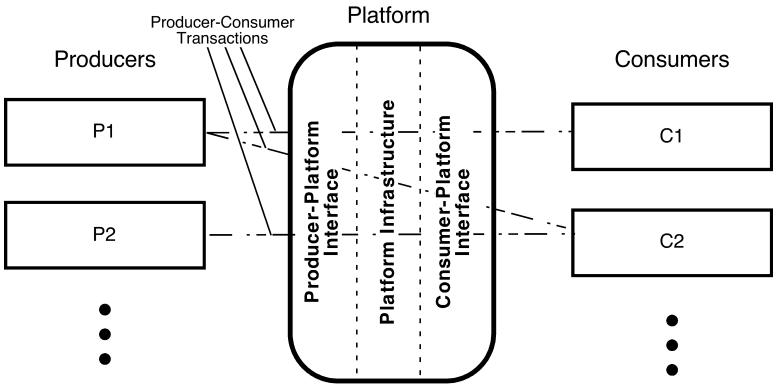


Figure 1
Two-sided Market

The following parts of a two-sided market are in Figure 1. (We'll use a credit-card platform for illustration.)

1. The Platform. Its three parts will be described below.
2. The Producers. (Vendors accepting the credit card.)
3. The Consumers. (Purchasers with accounts.)
4. The Producer-Platform Interface. The set of facilities provided by the platform that Producers must employ in order to participate. (Includes operating card readers, window decals, and payment services.)

Twitter: @conways_law
Date of pdf: 2019-01-02
conway.mel@gmail.com

5. The Consumer-Platform interface. The set of facilities provided by the platform that Consumers must employ in order to participate. (A credit-card and related account, for example.)
6. The Platform Infrastructure. (Includes payment-transfer, logging, and reporting services.)
7. The Producer-Consumer transactions between the participants, through each of which each participant derives value. (Credit-card purchases.)

A stable market exists due to network effects within the two populations that provide incentives to participate. Often the network must be seeded on one side first; this then draws participants on the other side.

A "killer app" can act as a nucleus that seeds the positive feedback loop. My recollection: American Express seeded the credit card market with business travel purchases: initially, airline tickets, hotels, and restaurants.

In the case of the PC software application market, these are the relevant parts.

1. The Platform is the PC with its operating system.
2. The Producer is an offerer of a software application product.
3. The Consumer is a potential application user.
4. The Producer-Platform interface is the operating system's API to which the application conforms.
5. The Consumer-Platform interface is the set of user-interface and other data exchange services provided by the operating system, as used by the users.
6. The Platform Infrastructure is the set of standards that insures compatible interchangeability of PC hardware and software.
7. The Transactions are the purchases/licenses of the applications.

The Microprocessor Revolution

Now let's go back to the microprocessor story. So something big happened in the early 1970s[14]. The semiconductor industry's decision to craft a maturing semiconductor chip technology in a new direction, a software-based computer architecture, led to two industrial paradigm shifts:

- a revolution in how manufactured equipment is controlled, from mechanical, hydraulic, and electromechanical to software, and
- a new economic activity: development of computer software applications for independent multiple deployments at consumer scale.

Each of these created its own two-sided market.

> Intel's decision to build a general-purpose programmable device wasn't obvious at the time. There is an anecdote[15] that a proposal made to Robert Noyce[16], a cofounder and executive of Intel, to build a processor chip for a desktop personal computer was met with resistance. The basis of Noyce's resistance was that each such computer would have only one processor chip, while it could have many memory chips, and Intel was in the memory-chip business. What Noyce may be forgiven for not having seen at the time was that this new device would lead to a whole new category and scale of economic activity. This new economic activity ultimately pulled Intel out of the increasingly competitive low-margin memory-chip business into leadership in the microprocessor business.

Enablers for Technology-driven Two-sided Market Creation

Here the emphasis shifts from a singular focus on inventions ("artifacts") to a dual focus on artifacts and on the communities of builders ("artisans") from which the artifacts emerge.

In my view the microprocessor revolution arose from the coincidence of three qualitative changes, enumerated here.

1. The invention of an interface mediating the collaboration of two communities previously unable to collaborate

The computer instruction set is an interface between two communities.

John von Neumann's invention of a digital computer whose control program is data stored in the same memory as the data being operated on[17] changed by orders of magnitude the ease of interoperation of technologies understood within two distinct communites:

- people who understood digital electronics ("hardware"), and
- people who understood discrete mathematics ("software").

Before this invention these two communities did not collaborate because they spoke the languages of different disciplines. Thus, where there was previously no way for the artifacts of these two communities to interoperate, the invention of an instruction set that could be instantiated as data in memory of the computer became an interface not only between their respective artifacts but between the communities themselves, enabling the construction of more complex systems combining hardware and software.[18]

2. A consequent business restructuring

The invention of the instruction set as interface induced the creation of two distinct business activities:

- manufacturing stored-program computers with standardized instruction sets, and
- writing software programs to be loaded into the memories of these computers[19].

3. An economic or technological shift that greatly enlarged the number of particiants

By 1970 Moore's Law[20] had progressed to the point that programmable digital computers could be put on monolithic silicon chipsets, and systems based on these chipsets could be built at costs enabling their sales to consumers.

Wiring is a *Connection*, Not a *Programming*, Language

Here we start building the case that the designs of interactive computer systems serving transaction-oriented businesses can be partitioned into two parts: a business-object-knowledge-intensive part that embodies the data and rules specific to the business, and a use-case-knowledge-intensive part that embodies the interactions between the system and its human users.

This section describes a departure in my thinking about business applications that occurred in the 1980s. Setting it out here can help in two ways:

1. It describes the reasoning leading to the "two-faced model" I have described elsewhere and will revisit below.
2. It helps to motivate the wiring model as the favored representation of code-free business use-case interactions.
   I view wiring not as an alternative programming language but as ***a static expression of a set of*** *(sometimes bidirectional)* ***connections*** *between business data and projections of those data on a user interface. This is what an editor is.*

Consider a typical interactive business application, for example in distribution. It is, essentially, an active intermediary between the business objects that carry the state of the business and the mechanisms that control the interactions with users.

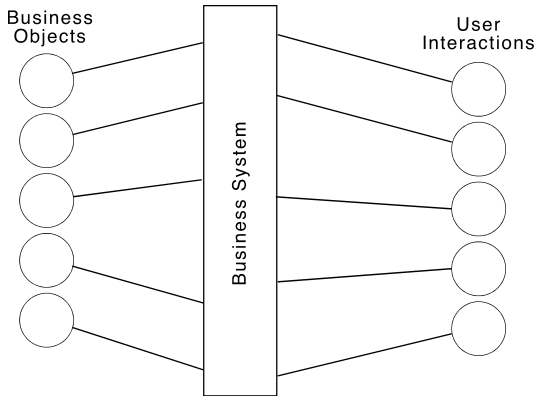


Figure 2
Simple Model of Interactive Business Application

Twitter: @conways_law
Date of pdf: 2019-01-02
conway.mel@gmail.com

Corresponding to the ways that businesses organize their workers, these user interactions can be grouped by department, for example, ordering, billing, shipping, receiving, payroll, etc., and each department contains a collection of department-specific use cases. At any moment, each interaction is engaged in the realization of one use case:

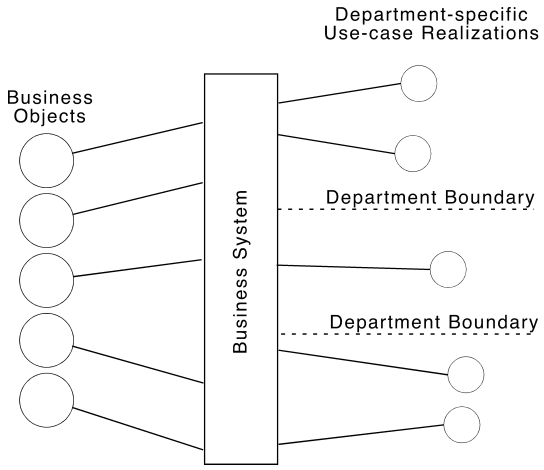


Figure 3

Use Case Realizations are Business-object Editors

On the surface of each user interface are (possibly user-changeable) projections of parts of the business objects. Thus we have:

The Editor Hypothesis

- *Every interactive business system is a collection of interactive use-case realizations.*
- *Each such interactive use-case realization is a business-object editor.*

This is the organizing principle that led to the wiring model.

Now reconsider the two-faced application model[21]. Figure 4 here changes the language a bit to conform to the current discussion.

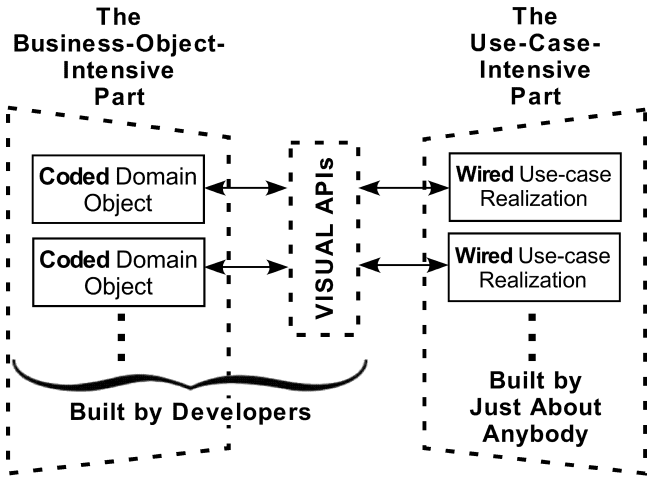


Figure 4
The Two-faced Application Model

Note the structural similarity to the two-sided market shown in Figure 1. This is no accident. Here is the correspondence:

Two-faced application	Two-sided market
Coded domain object	Producer
Wired use-case realization	Consumer
Visual APIs[22]	Platform
Service calls from wired gateway components to domain objects	Producer-Consumer Transactions

There is a subtle inconsistency here, though: on the right the Producer and Consumer are people. On the left the domain objects and use-case realizations are software.

This inconsistency is our opening to distinguish between the two phases of the life cycle of a product:

- The *design* phase, during which the *artisans* are active, and
- The *operation* phase, during which the *artifacts* are active.

The way we resolve this distinction into a single concept is to elaborate on the two-sided market diagram in Figure 1, showing in Figure 5 below both life-cycle phases. In the process we will choose some suggestive names, revealing the structure of a platform-based two-sided market in the design phase.

What might not be obvious to some readers is that *both phases occur together side by side* in the wiring tool because the tool conforms to the Humane Dozen[23].

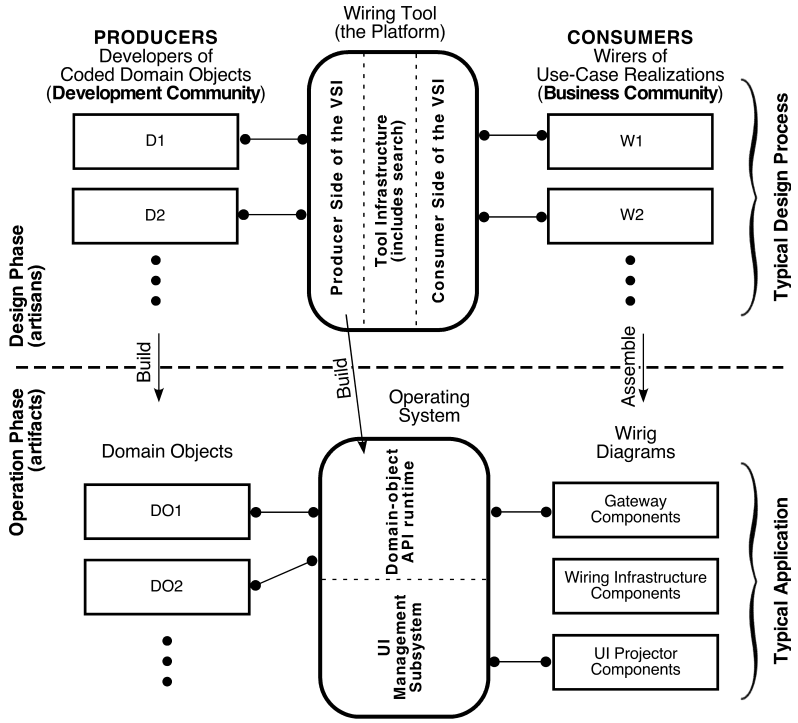


Figure 5

Wiring-platform-based Application Development Process

Can the creation of value in the form of executable wiring diagrams that employ services from domain objects be the basis of a viable two-sided market? We'll address this question next.

Creation of a New Two-sided Market

The platform of Figure 5 satisfies the list of enablers derived from the history of the microprocessor as the basis of a new technology-based two-sided market.

Here are the enablers of a two-sided market created by the emergence of a new technology; we derived these before from the history of the microprocessor.

1. The invention of an interface mediating the collaboration of two communities previously unable to collaborate

The two communities are software developers (primarily concerned with the technology) and business people (primarily concerned with economic activity potentially supported by computer technology). The latter are typically non-programmers. The business people create executable wiring diagrams in a code-free drag-and-drop wiring tool. The wiring diagrams consist of components dragged out from component palettes; these components have source and sink connectors, and the wirers can draw wires between the connectors. Each wire denotes a flow of an object from a source to a sink. Some of these objects can be domain objects obtained from Gateway Components.

Some Gateway Components accept Domain Objects at their sink connectors. On the wirer's request, an input Domain Object to a Gateway Component (using the services of the VSI) will reveal its API in a dialog-based form that I call the Domain Object's *dashboard*.[24]

The platform includes the two-sided VSI definition whereby domain objects built by developers using their own tools can be accessed by Gateway Components to:

1. Display the input domain object's dashboard, and
2. Build the operation-phase service request to the domain object.

2. A consequent business restructuring

If the platform is built and managed correctly, network effects will induce developers to build domain objects and business people to build wiring-diagram-based applications. Freemium pricing arrangements and an initial inventory of free domain objects and wired components (yet to be defined) will induce business people to build things they want and developers with specific domain knowledge to embody that knowledge in domain objects.

Twitter: @conways_law
Date of pdf: 2019-01-02
conway.mel@gmail.com

3. An economic or technological shift that greatly enlarged the number of particiants

The two-part application model partitions the application-building process in such a way that the code-free wiring part will be accessible to almost everyone. From my research on simplification I believe that this part can be made close to universally accessible.[25] It is an appropriate model for business people and others to build descriptions of user interactions, which is a natural starting point when experimentally building an application. The Gateway Component is wired like every other component and offers a low-friction connection to the more sophisticated services offered by domain objects.

Consumers Can Also Be Producers

The platform has the ability to encapsulate a wiring diagram, turn the result into a wired component, and add that to a component palette; I have demonstrated the feasibility of this. Wired components produced this way will be indistinguishable to the consumer from wired components produced by coding.

In addition to encouraging consumers to become producers, this ability supports consumer-side network effects, for example among business people collaborating on an application. These people need not be in the same organization but might be in an affinity group; thus the platform could be managed to support an enlargement and partitioning of the open-source process to include non-programmers building wiring diagrams for the library.

Growing the Platform

This section envisions the life cycle of the platform, from a tenuous beginning to a stable state.

I visualize managing the platform to conform to Christensen's model of disruption[26]. At this time I imagine three stages of the platform's life cycle. These are discussed in the following subsections: early adopters, vertical markets, and democratizing application development.

Early adopters

This stage begins with a minimum viable platform that is valuable to one or more communities of early adopters. A tight feedback loop between the platform and its users grows the platform in directions suggested by the usage patterns.

At this stage pricing is free, but the intention to introduce freemium pricing for the benefit of owners of software that can be converted to proprietary domain objects should be stated, in order to begin conversations with these owners.

Wrapped office applications. Horizontal office applications such as a document editor, a spreadsheet, and a relational database[27] should be wrapped in order to make them accessible in the platform as sources of business objects to Gateway Components. Ideally a system such as Apache OpenOffice will have an API[28] that can be encapsulated for this purpose.

Initial consumers will include enthusiasts, educators, experimenters, and opportunistic system integrators who see the platform as a rapid-application-development tool.

Internet of Things. The standardized VSI presents the opportunity to build cross-manufacturer IoT applications. I see an opportunity to wrap the drivers of a variety of widely-used devices as domain objects and offer a control-panel builder that integrates the devices of different manufacturers.

Vertical markets

There is a large vertical-industry software business[29,30] that has captured and mechanized in application software knowledge of numerous market segments. I posit here that repackaging this knowledge in domain objects in the proposed platform is a more productive repository for this knowledge because it frees the knowledge from specific applications, making it

reusable in multiple use cases. This raises the unanswered question: how to provide incentives for the people with the knowledge to liberate it in this way. Part of such an incentive structure would be pricing that compensates for the loss of application sales and offers an enlarged market because of the broader utility of the repackaged knowledge.

There is also the option to choose a promising vertical industry and contract with an expert in that industry to build seed domain objects.

We might find that some vertical markets are served by system integrators that mediate between vertical-industry-software vendors and end users. Repackaging industry knowledge as domain objects in the platform can empower these system integrators to enlarge their offerings, customize them, and mix offerings from different manufacturers.

Democratizing Application Development

In the longer run I see some descendant of this platform or something similar restructuring application-development practitioners into two communities: producers (domain-object builders) and consumers (both programmers and non-programmers). This will move us one step closer to universal accessibility of software understanding.

---

[1] https://en.wikipedia.org/wiki/Transistor-transistor_logic

[2] https://en.wikipedia.org/wiki/Intel_4004

[3] This delayed-binding-as-enabler meme shows up in many different places and deserves its own theory. As a placeholder, consider http://melconway.com/talks/2018_gotober/ .

[4] The Lockheed-Martin F-35 Joint Strike Fighter is the poster child of this trend: "Maintenance personnel have discovered that it is often possible to correct deficiencies in the F-35, *which is a software-defined aircraft*, simply by rebooting the aircraft's software and onboard systems." (Emphasis mine.) https://en.wikipedia.org/wiki/Lockheed_Martin_F-35_Lightning_II

[5] https://en.wikipedia.org/wiki/MOS_Technology_6502

[6] https://en.wikipedia.org/wiki/Motorola_68000

[7] https://history-computer.com/ModernComputer/Software/Visicalc.html

[8] http://web.mit.edu/shabby/www/nytos2.html

[9] https://www.statista.com/statistics/218089/global-market-share-of-windows-7/

[10] Clayton M. Christensen (original publication 1997), *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. Harvard Business Review Press

[11] https://www.newyorker.com/magazine/2012/05/14/when-giants-fail

[12] https://en.wikipedia.org/wiki/Two-sided_market

[13] G. Parker, M. Van Alstyne, S. Choudary (2016), *Platform Revolution: How Networked Markets Are Transforming the Economy - and How to Make Them Work for You*. New York: W. W. Norton

[14] Something else big happened then. As you read on you might suspect with me that there is more going on than mere temporal coincidence: https://www.epi.org/productivity-pay-gap/

[15] https://history-computer.com/ModernComputer/Basis/microprocessor.html

[16] https://en.wikipedia.org/wiki/Robert_Noyce

[17] https://en.wikipedia.org/wiki/Stored-program_computer

[18] Von Neumann's original justification for putting control instructions in memory was a technical one: doing so permitted creating programs whose behavior was data-sensitive, for example convergent iterations. I assert that today the principal justification has changed to an economic one: delayed binding, which is what has enabled the software business.

[19] However, it took an antitrust lawsuit to break IBM's cross-subsidization that had been blocking this:
https://en.wikipedia.org/wiki/History_of_IBM#1969:_Antitrust,_the_Unbundling_of_software_and_services

[20] https://en.wikipedia.org/wiki/Moore%27s_law

[21] http://melconway.com/talks/2018_gotober/09.html

[22] http://melconway.com/talks/2018_gotober/10.html contains an informal description. The developer of a service call writes a single method for each service that, when called by a gateway component, opens a sequence of dialogs, also built by the developer.

[23] http://melconway.com/Home/pdf/humanedozen.pdf

[24] http://melconway.com/talks/2018_gotober/21.html shows an example of summing the items in a shopping cart.

[25] I have demonstrated a wiring tool that fulfills 11 of the 12 principles of http://melconway.com/Home/pdf/humanedozen.pdf , and the 12th, Undo, will be straightforward.

[26] Clayton M. Christensen, *op. cit.*

[27] In http://melconway.com/talks/2018_gotober/17.html I show what the dashboard of the SQL Select verb might look like.

[28] http://www.openoffice.org/api/

[29] Estimated by Bowery Capital at $122B: https://bowerycap.com/blog/insights/vertical-market-software/

[30] A suggestive list of vertical markets is at https://www.g2crowd.com/categories/vertical-industry