The Case for a FOO-technology Web-based Application Development Service

Mel Conway

If there is a technology that can radically increase productivity building applications to run in Web pages, smart phones, and other GUI devices, including those in the new iPad category and ordinary client-server computer applications, then there will appear a large number of new developers to generate these applications.

Here is a development technology that promises to create this new market of application builders including programmers as well as a large new number of wannabe programmers and non-programmers. This market can be an order of magnitude larger than existing developer markets based on conventional programming tools.

The technology exists, is well documented, and can be demonstrated in the form of a substantial proof-of-concept prototype.

Abstract: FOO (Flow Objects Online) is a graphical application development technology that can radically improve the rate of developing server-based end-user applications, compared to conventional text-based programming. Its use is not inherently restricted, but it stands alone in the field of technologies for rapid development of end-user applications that run on remote servers.

An appendix makes the case that the application of the FOO technology in support of low- and moderate-skill developers and wannabe developers can radically enlarge the market for a Web-deployed application development service, compared to the market for a similar service based on conventional text-based programming tools.

The argument has three parts.

1. A premise: A web-based application development service such as that described on page 2 of this paper.
2. A premise: The FOO graphical development technology can indeed multiply the productivity of ordinary developers of end-user applications by an order of magnitude. Page 3 of this paper briefly describes the technology.
3. A statement that, under reasonable assumptions, the increase in the size of the market of the development service varies as the *square* of the productivity factor (the factor that FOO multiplies development productivity over conventional programming). The remainder of the paper shows the justification of this statement.

conway.mel@gmail.com

Rationale for the service description below: It will be shown on the following pages that the potential size of the market for a software development technology is strongly influenced by the general level of productivity (amount of useful function created per unit time) afforded by the technology. Given the usual power-law distribution for productivity in a population of developers, adopting a technology that increases the general productivity level by a factor enlarges the number of economically productive developers by the square of that factor.

The amount that a development technology facilitates each developer's reuse of other developers' code to a depth of several levels is the single most powerful influencer of productivity, other things being equal. Several object-oriented textual programming languages facilitate such reuse well. They form a mature technology whose inherent productivity characteristics are understood.

Graphical languages can be simpler to use than textual languages, so they potentially open up the development tool market to a larger group, including some who today cannot program. But graphical languages have suffered from three major shortcomings: (1)They do not generally obviate the need for textual coding or scripting; (2)A given graphical language is typically not applicable to a broad range of application types; (3)Most critically, they have not been shown to support multi-level reuse.

The graphical technology described here can create the entire class of interactive GUI applications, and it is unique in its support of multi-level reuse. Because it combines the cognitive accessibility of a visual programming model with multi-level reuse it promises to create a large new population of economically productive developers.

Build Applications Over the Web Without Coding

Development Is Entirely Graphical	Both programmers and non-programmers build business applications using a visual "component-wiring" or "flow" language. Simple applications are graphical and easy to build. Complex applications are complex, but still graphical and still possible to build. All aspects of an application, from the database to the user interface in the browser, are built by wiring up components.
There Is No User Coding	There is a stable set of "primitive" components built by expert programmers from which all other components are built by encapsulation of wiring diagrams. Given a mature set of primitive components, variations needed for specific applications are obtained by option selection in dialogs, not by code.
Simplicity Raises Productivity	The flow model is more understandable than programming, yet generality, scalability, and application performance are not sacrificed.
Collaboration Raises Productivity	Users assemble "open source" wired components from a public component library. Users create reusable wiring diagrams from library components, encapsulate them, and put new components into the library for others to use. Network effects raise the general productivity level.
More Productivity Means More Developers	A substantial increase in productivity implies an order-of-magnitude increase in the developer population.
Development and Deployment Are Worldwide	Developers create and edit wiring diagrams within Web browsers. The applications execute from the servers of the development service.

conway.mel@gmail.com

Brief Description of the FOO Technology

What kind of a development technology can increase the productivity by an order of magnitude over conventional programming of a whole community of end-user application builders?

The FOO Technology

The FOO technology has the following properties.

- The development language is not a conventional textual language but employs an intuitive graphical process of assembly of visual components by "wiring" them together. Data flows over the wires from database to user interface. Concrete flow processes can be understood by more people than textual processes.
- The user wires together components from the library and can create components for reuse by others. There is a way to generalize a wiring diagram, encapsulate it, and add the encapsulation back to the library. As part of the Web-based development service, this reuse process takes advantage of network effects to raise the general productivity level.
- Components created by encapsulation are indistinguishable from components created by conventional programming, insofar as they are employed in the assembly process. This is a property of all successfully extensible programming languages.
- An application is simply a special case of a component created and added to the library, one that runs by itself.

My experience as a user of FOO suggests that an order-of-magnitude increase in rate of development, compared to textual programming, is within reach.

The Dataflow Application Model

The combination of the following two properties is what affords a radical increase in development productivity:

- Multiple levels of reuse.
- An intuitive, concrete graphical assembly process.

In order for these two properties to coexist comfortably, a dataflow application model is necessary. FOO employs such a dataflow model. It has specific technical characteristics that are successfully hidden from the general population of application developers; this successful hiding is what makes the component wiring process look simple. Of course, the details of the technology have to be somewhere, and in FOO they are in the assembly tool and the primitive components, i.e., the bottom-level components that are built by conventional programming and from which all other components are built. The primitive components and the assembly tool must be built by programmers who are more expert than those application developers (and wannabe application developers) in the general population addressed by the Web-based application development service.

conway.mel@gmail.com

Appendix

Defense of the Statement that the Size of the Market for the Web-based Development Service Varies as More Than the Square of the Productivity Increase

The Power-Law Productivity Model

Defining productivity as amount of useful function created per unit of time, let us pursue the implications of the general observation that programming productivities vary widely. There is a small number of superstars, a large number of programmers with average productivity, and an even larger number of programmers (and wannabe-programmers) with below-average productivity. In the following discussion we consider the entire pool of developers and potential developers.

We assume, with some justification from experience, that the productivities of the individuals in this pool are distributed according to a Power Law distribution. [See, for example, Shirky: http://www.shirky.com/writings/powerlaw_weblog.html .]

The power law curve is, in its simplest form, the $y = 1/x$ curve. We need a specific function that describes the productivity distribution of our pool of developers and wannabe developers. We start by defining that an individual has "acceptable productivity" if that individual can develop an arbitrarily defined standard application in a period of time that is less than or equal to an arbitrarily-determined fixed amount of time. (For example, we could define acceptable productivity as being able to build a standard application in six months or less.)

Our power-law assumption reads like this. Consider the entire pool of developers and wannabe developers, those with both acceptable and less-than-acceptable productivities. Choose a random group of N individuals from the population, where (1)all N individuals have acceptable productivity, (2)the least productive of these has exactly the minimum acceptable level of productivity, and (3)N is small compared to the total size of the pool. Ranking these N in order of decreasing productivity with the most productive at position number 1 and the least productive at position N, we assume that *each of the top 1/5 of these N developers has a productivity at least twice the minimum acceptable productivity*. An informal way of saying this is: if the productivity of the developer with rank n is P, then the productivity of the developer with rank $n/5$ is $2P$.[1] That assumption gives us the function we are looking for, as follows.

[1] The work performed by the first m individuals in a given amount of time is proportional to the area under the power curve from 1 to m. Under the assumption being made here the first 20% of a given developer population does 28% of the work of the total population. This is a much weaker assumption than the 80/20 rule.

conway.mel@gmail.com

Order *N* developers inversely by productivity, with the most productive first. Assume all *N* can build at least *a* standard applications in the fixed time period, and the *N*th developer can build exactly *a* standard applications is that time period. We assume a general power-law productivity relationship in which the developer in position *N* can build *a* applications and the developer in position *N*/5 can build 2*a* applications. If the developer in position *n* can build *A(n)* applications, then there are constants *k* and *K* such that

$$A(n) = K \cdot n^{-k}$$

and we solve for *k* by substituting

$$a = K \cdot N^{-k}$$
$$2a = K \cdot \left(\frac{N}{5}\right)^{-k},$$

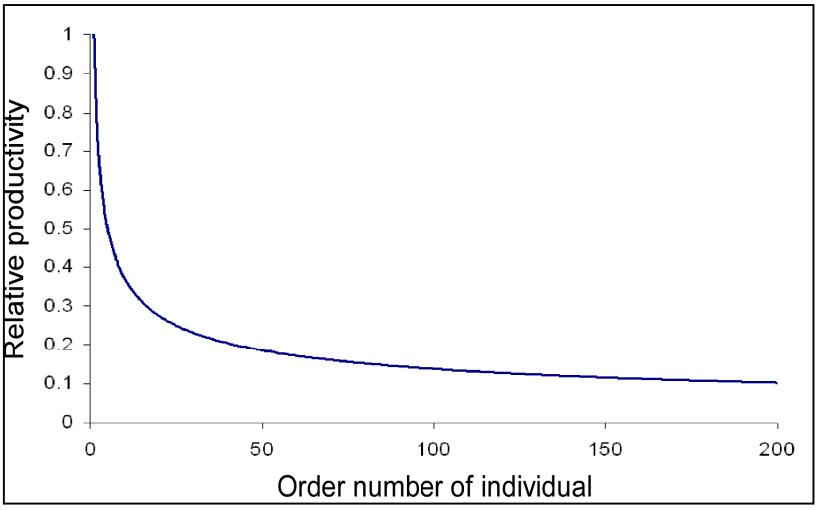
which gives

$$A(n) = K \cdot n^{-0.431}.$$

We define Relative Productivity *P(n)* of developer *n* by scaling the curve so that the productivity of the most productive developer in the population is 1.0

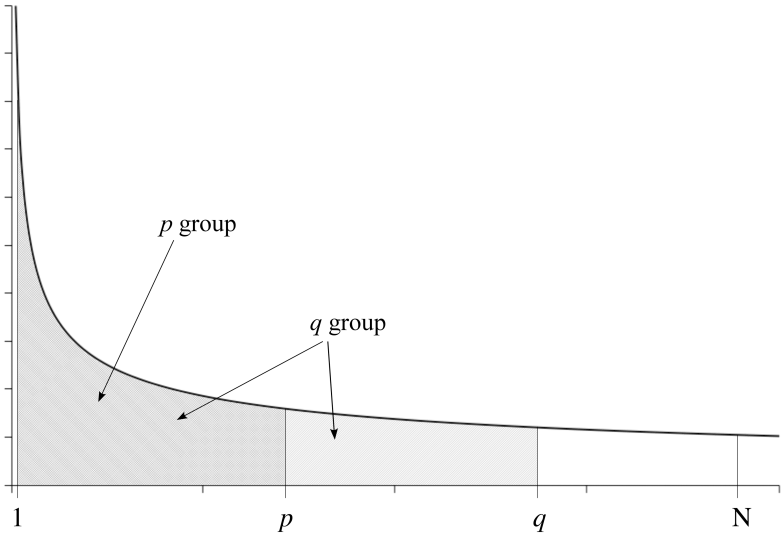
$$P(n) = n^{-0.431}$$

This figure shows the shape of the function.



conway.mel@gmail.com

Consider the group of the top p individuals in the population, and the larger group of the top q individuals in the same population, where $p < q \leq N$. The p group is a subset of the q group, and they both contain the most productive individuals numbered 1 to p in the total subpopulation.



The ratio of the productivity of individual q to that of individual p is less than 1 and is given by the ratio

$$\frac{q^{-0.431}}{p^{-0.431}} = \left(\frac{q}{p}\right)^{-0.431}$$

For the sake of illustration let's assume that individual p is 10 times more productive than individual q. Therefore

$$\left(\frac{q}{p}\right)^{-0.431} = \frac{1}{10}$$

which gives

$$0.431 \cdot \log\left(\frac{q}{p}\right) = \log 10$$

and

$$\frac{q}{p} = 10^{2.32} \approx 209 \ .$$

conway.mel@gmail.com

This interesting result says that when the productivity of individual *p* is 10 times the productivity of individual *q*, the size of the *q* group is over 200 times the size of the *p* group.

We can generalize this result to a productivity ratio of *R* instead of 10. If

$$\left(\frac{\text{productivity of individual } p}{\text{productivity of individual } q}\right) = R$$

then

$$\frac{q}{p} = R^{2.32} .$$

That is, the size of the *q* group is more than R^2 times the size of the *p* group.

Size of the Population of Useful Developers is Very Sensitive to Productivity

Using this result, assume that at time 1 the state of the development art is that the productivity of developer *p* is the minimum useful productivity. (Therefore, individual *q* has less-than-acceptable productivity.) Then, at time 2 the technology has changed and everybody's productivity is raised by a factor *R*. At time 2, then, developer *q* is the developer with the minimum useful productivity. From time 1 to time 2 the size of the population of developers with useful productivity is then raised by a factor of *q*/*p*, or more than R^2.

This effect might not be noticed for small productivity increases, but *a radical productivity increase can reshape the developer population*.

Based on years of experience as a user of a prototype FOO-based development tool, and assuming a tool mature enough to build a broad collection of applications, I believe that a productivity multiplier of 3 over conventional programming is a conservative estimate. Therefore,

> **The market for a FOO-based application development service can be an order of magnitude larger than the market for a similar service based on conventional programming.**

conway.mel@gmail.com

The figure below demonstrates graphically why the shape of the power curve causes this important effect. It shows the qualitative difference in the number of acceptably productive developers based on the earlier assumption of a productivity multiplier of 10. Let the bottom curve represent the productivity distribution using an old technology, and let the top curve represent the productivity distribution using a new technology that multiplies everybody's productivity by 10. The curve shows individuals 1 to 500, ranked in order of decreasing productivity. (A relative productivity level of 1 is defined to be the productivity of the most productive programmer using the old technology.) Now assume that 0.74 is the minimum acceptable productivity level. With the old technology there are only two individuals with acceptable productivity. With the new technology there are 418 individuals with acceptable productivity.