1. Introduction

Since the appearance of general-purpose computers, the question of how best to organize and manage the creation of machine instructions—how best to develop software—has remained open and important. The advent of English-like “3rd Generation Languages” (3GLs) increased development productivity. The number of people writing software, the volume of software written, and the average size of systems increased. Work formerly undertaken by individuals evolved into complexly organized group activity (Brooks, 1975). Schools of thought have emerged to advocate one development process or another; debates about how to manage software developers and their work fill meetings, journals, and books (e.g., Beck and Boehm, 2003).

This paper proposes a model that relates (1) the demand for novelty in a product and (2) the underlying cost structure of production tasks, to (3) the way in which development should be structured. In devising the model, we apply economic reasoning to questions implicit in the “methodology debate” (Highsmith, 2001) between developers who emphasize planning before coding and those who emphasize going early to code prototypes. The model specifies conditions in which developers should use each approach, and suggests that both sides in the methodology debates may be correct, depending on the characteristics of the environments in which they work. We frame our arguments within a discussion of how software development and other productive activities have changed historically, and intend to derive explanations that apply to productive activities beyond software development. Framed in historical context, the model suggests that the emergence of “agile” software development might exemplify a more general transformation in the nature of work. We note similarities among agile development, evolving approaches to other kinds of knowledge work, and long-
established practices in the collaborative arts, based on similarities in circumstances and underlying cost structures.

2. A (Very) Brief History of Software Development

While remaining aware of the imprecision necessary to any brief history of software development (and of the opportunities for giving offense to advocates of one method or another), we suggest three categories of approach to development process. These correspond roughly to periods in the history of computing. We call the three categories, ad hoc, planning intensive, and agile; we characterize them as follows.

Ad hoc development cedes design and management of development process entirely to the programmer; the ad hoc developer is a craftsman, guided by personal expertise and methods. An ad hoc process can assume any shape that produces machine-runnable code that performs useful tasks. This category of development is characterized, therefore, by lack of imposed process structure. Of course, a process that can take any shape can produce software of any shape, within the limits of syntax and functionality required to create running and useful software; such wide-ranging product heterogeneity poses maintenance, improvement, quality assurance, and other management challenges. For this reason, ad hoc development is most often associated with the “bad old days” of software development, before any structure was imposed on developers’ work. An ad hoc development process, according to a common criticism, is no process at all.

Planning intensive approaches arose to address the perceived shortcomings of ad hoc development. Critics of the ad hoc approach sought to impose structure, first on the shape of software, then on the process for developing it. Advocates of “structured programming” (Bohm and Jacopini, 1966; Dijkstra, 1968; Dahl, et al. 1972; Knuth, 1974)
proposed restricting the structures written into code (the machine-readable instructions) to a small subset of possible structures (e.g., sequences, decisions, and loops), to make software easier to write and change. Such restrictions did not limit functionality, but did improve productivity, and the ease with which programmers could work on each other’s code or take up code again after putting it aside a long time. Object-oriented programming (Dahl and Nygaard, 1967) extended the idea of imposing standardized structure on code so that applications could be built from basic components, thus achieving further productivity and accessibility gains.

Efforts to apply structure to the process of creating software followed. Royce (1970) proposed an influential stage-structure model, known as the “waterfall model.” In its most common form, this model describes a process that moves a product from abstraction to physical detail in stages, beginning with requirements documentation, then moving to analysis, design, implementation, test, and software deployment (see Figure 1).

Figure 1: The Royce “Waterfall Model”

Efforts to apply structure to the process of creating software followed. Royce (1970) proposed an influential stage-structure model, known as the “waterfall model.” In its most common form, this model describes a process that moves a product from abstraction to physical detail in stages, beginning with requirements documentation, then moving to analysis, design, implementation, test, and software deployment (see Figure 1).

1 Calls for codification of the software process itself reached what was, perhaps, an apex when Osterweil suggested, in the title of a 1987 paper, “software processes are software too.”
Developers on a waterfall project create representations (i.e., documents) that specify the features a product will need, then develop these specifications in stages of increasing detail until they become code-like (pseudo-code) and, finally, actual code. The work moves to each new stage after having satisfied thoroughness and other quality criteria. Royce envisioned iteration backwards between successive stages, as well as movement forward, but recommended steps that would prevent the need to iterate back more than one stage.²

Boehm (1981) analyzed software development at a large company and concluded that the cost of making changes to a product rises exponentially as it moves toward completion. That is, changes needed in later stages of the waterfall are many times more expensive than changes noted and made earlier (see Figure 2). This cost

---
² A fair reading of the original paper reveals that Royce never used the expression “waterfall” and that he devoted a great deal of discussion to the issue of iteration between stages. But his model is sequential in nature, and his recommendations are aimed largely at preserving that nature.
structure supplies a rationale for the sequential nature of a waterfall approach, a reason to be thorough in one stage before moving to the next. Moreover, Boehm’s observations and the waterfall model both closely resemble similar descriptions in other project-related engineering disciplines (e.g. Clark and Fujimoto’s, 1993 description of automobile manufacturing; Figure 3), a fact which lends to them strong intuitive appeal. Planning intensive models have been widely influential and continue to hold sway in many modern contexts (Walton, 2004).

![Figure 3: Clark and Fujimoto’s Depiction of the Automaking Process](image)

But planning intensive models have numerous detractors, especially in recent years. One common criticism of such approaches is that they proceed too slowly, so that even when successful on their own terms, they produce solutions to problems that have already changed. Critics also doubt that software requirements can be discerned well enough in advance to serve as sequential inputs to the development process; no requirements gathering stage can be thorough enough, they argue, to prevent substantial changes in what is required of a system as it is developed. Some
requirements will be missed and many that are not missed will change. Thus, they conclude, the idea of a coherent, ex ante discernible set of requirements is a misleading and potentially dangerous fiction.

Supporters of planning intensive approaches (including Royce himself) acknowledge the need to incorporate unforeseeable changes and have proposed variants of the basic model; for example, the “Spiral Model” (Boehm, 1988) loops through each stage more than once, coming ever closer to the needed solution as the process “spirals inward.” The grandly titled and influential “Unified Software Development Process” (Jacobson, et al, 1999) combines formal methods for representing requirements with iterative refinement of those requirements. Indeed, many modern planning intensive approaches add feedback loops and some overlapping of stages to make processes less sequential.

Radical critics of planning intensive approaches (see, for example, Beck, 2000) go further. They argue that exponential cost increase is not a necessary characteristic of software development *per se*, that the cost curve described by Boehm is a consequence of the design of planning intensive approaches, and that other process designs are possible. *Agile* developers claim to design processes in which costs of making changes do *not* escalate exponentially; this flattens the Boehm cost curve (see Figure 4). The idea has far reaching implications. It weakens the rationale for separating projects into stages. Rapidly generated prototypes—going quickly into coding—evokes feedback about system requirements and thus substitutes for thoroughness in planning. Requirements are uncovered in iterative discussion of prototypes, rather than elicited from interviews about prospective features of hypothetical future systems.

Figure 5 depicts an agile process used by Trilogy, a Texas-based software development firm (Austin, 1999). The “Manifesto for Agile Software Development” (agilealliance.org) sets out characteristics shared by agile approaches: 1) cheap and
rapid iteration; 2) “early and continuous delivery of valuable software”; and 3) a welcoming attitude toward changing requirements, even late in development. The Manifesto also welcomes people-centered management and urges leaders to “build projects around motivated individuals,” “give them the environment and support they need,” and “trust them to get the job done.” Beck describes the software development process as “conversation” (2001) and emphasizes the benefits of collaboration (e.g., “pair programming”) and “sustainable pace” (as opposed to long hours and overtime). This approach privileges the human element in software production; agile programmers insist that their work cannot be Taylorized to standard and interchangeable assembly line jobs (Beck, 2001). As in ad hoc development, the capabilities and characteristics of the programmer (especially the ability to quickly create prototypes for discussion) return to the fore, casting him or her as more an “artist” (or, “professional,” at least) and less an interchangeable worker within a standardized production system.

Figure 4: The Flattened Cost Curve Suggest by Agile Developers
Here lies the point of greatest philosophical disagreement: To some, returning the programmer to the fore sounds like a step backward, away from science (or engineering). They envision a future in which software development, like many other productive activities, will become increasingly rationalized, standardized, and predictable. To supporters of this view, agile approaches appear to be ad hoc approaches dressed in more respectable clothes (Highsmith, 2002).

![Diagram of Agile Development Process at Trilogy]

Figure 5: Agile Development Process at Trilogy

Empirical evidence shows that commercial software companies reacting to time-to-market demands avoid planning intensive approaches and instead adopt practices like those of agile developers (Iansiti and MacCormack, 1997). Further, the emergence of agile approaches has corresponded with changes in the computer industry; Baldwin and Clark (2000) demonstrate that distribution of market capitalization in the industry has changed from concentration in one firm (in the 70s) to a more fragmented structure in which firms are more often selling software directly into a competitive market (in the 80s, 90s and beyond). Depending on your point of view, the rise of agile methods demonstrates either a triumph of bad practice driven by short-term business interests, or
the emergence of superior methods from the discipline imposed by the market. The model developed in this paper turns light upon this disagreement.

Indeed, while we are interested in software development for its own sake, we are most interested in how the relatively short history of this field parallels the long evolution of the nature of work generally. Software making is a microcosm that highlights a historical pattern. In deriving the model, we trace changes in productive activities from ancient through industrial and into knowledge-intensive making. These historical ways of working correspond closely to ad hoc, planning intensive, and agile approaches to creating software. As we will show, the changes taking place in the software development world may herald general changes in the nature of work as we move into a more information and knowledge intensive economic age.

3. A Theory of Making

For insight into the evolution of methods in software development, we step far back in time to consider the nature of work before the industrial revolution, when makers were expert craftsmen. Many have argued that software development struggles to move forward from this situation even today. To address this possibility, we construct a simple model that helps us understand important characteristics of work and how they evolve through time as support technologies and methods change.

3.1 Ancient Making

Consider the situation of a medieval blacksmith who has none of the ideas about making that we take for granted. He has no ideas about economies of scale, interchangeable parts, or standardized work processes. This man makes unique things, say knives, one
at a time, from variable materials. What he does in his work varies to accommodate differences in materials (individually and in combination) and the differences arising from the interdependence of his materials and methods. One batch of charcoal burns hotter than another; this load of ore has a greater or lesser iron content and makes a different bloom of metal than another, even from the same mine. These differences mandate differences in furnace time, in forging, in annealing and tempering, and inevitably in the final product. Each knife he makes is unique, and it never occurs to him that it could be otherwise.

Ancient making generated three kinds of cost: 1) the effort and resources required to arrange and rearrange equipment and materials (Keller and Keller, 1996), which we call reconfiguration cost and denote $C_R$; 2) the effort and resources lost when something did not work well enough to become part of the final product, which we call exploration cost and denote $C_E$; and, 3) the effort and resources consumed in making each part of the final product, which we call variable cost and denote $C_V$. For example, adjusting the fire to account for differences in iron content of ore incurs reconfiguration cost; recycling an incorrectly tempered blade that broke under testing incurs exploration cost; the work and materials used to make a saleable knife, independent of reconfiguration and exploration, combine to produce variable cost. The sum of these costs we call iteration cost:

$$C_I = C_R + C_E + C_V.$$ (1)

Note that the individual cost components, $C_R$, $C_E$, and $C_V$, are separable only in concept. For the ancient maker, they intermingled in ways that we, thinking with post-industrial revolution categories, find difficult to imagine. To the ancient maker,

---

3 To further illustrate the distinction between reconfiguration and exploration costs, we suggest a more modern example: If you want to make a different kind of airplane, you might rearrange your production process (e.g., the factory) to enable this. The costs of rearranging the plant would be reconfiguration costs. The cost incurred if the new kind of airplane crashes when test flown (in loss of equipment, life, etc.) is exploration cost.
reconfiguring and exploring was part of what had to happen every time he made a new knife. Differences in every aspect of his task—customer preferences, material and tool properties, flow in the millrace, his own performance—management of all of these constituted his expertise. No matter how many knives he made, no matter how similar some might be to each other, he would expect to incur all three costs for every knife. Even if he made a matched set of knives for a butcher, differences in materials, accumulation of differences in the outcomes of uncertain processes, and so on would require constant process adjustments: reconfiguration and exploration. Even if identical final products would suit the customer, the variations in the process of making—to account for different fire temperatures, mistakes made one time but not the next, etc.—incurred reconfiguration, exploration, and variable costs every time, and thus his expectations about the average cost of making a knife remained more or less constant no matter how many were made.

Because the ancient maker expected to incur all three costs each time he made something, the total cost he expected to incur in making multiple knives was

\[ C_{TOT} = C_i x \]  

(2)

where \( x \) is the number of knives produced; the average cost in making knives was,

\[ C_{AVG} = C_i. \]  

(3)

The benefits afforded by this maker’s product took a different form. Because the ancient maker’s product was for a particular person, it provided benefit \( B_i \) to that person, that we denote \( B_i \). It also provided some benefit to anyone who used it; a knife made for a larger hand might mean painful blisters in hard use, but it could cut and thus was better than nothing. This core value that the product provided for any user we denote \( B_c \). The total benefit created from making multiple knives was, therefore

\[ B_{TOT} = B_i + B_c x \]  

(4)
where \( x \) is the number of knives of the same kind produced; the average benefit in making knives was, therefore,

\[
B_{AVG} = \frac{B_I}{x} + B_C
\]  

(5)

Notice, the average benefit of knives that are alike, unlike average cost, is not constant. Rather, it falls and approaches the core value, \( B_c \), asymptotically as \( x \) increases. The first knife of a kind provides individual and core benefit to the person for whom it is designed; other knives like it provide only the core benefit to people for whom it is not designed; so average benefit approaches core benefit as more like knives are made.

If we define profit as benefit minus cost, we arrive at the following expression:

\[
? = B_{AVG} x - C_{AVG} x
\]

\[
= (\frac{B_I}{x} + B_C)x - C_I x
\]

(6)

Assume that parties to a transaction need a surplus to divide among themselves, thus that transactions occur only when \( ? > 0 \). If we set aside of the uninteresting case in which transactions are always profitable, i.e., \( B_c > C_I \), (in which core benefit is greater than cost of iteration), we are left with a more interesting case in which transactions will occur:

\[
B_c < C_I \text{ and } B_i + B_C > C_I
\]

(7)

Figure 6 shows graphically the relationship between benefits and costs in this case. Note that transactions are possible only when small numbers of like items are made. In this ancient making world, transactions usually occur when a single customer places an order for a single, unique product. The product has benefit to that individual that exceeds its rather high cost, but the product made to fit for that customer is not useful enough to other customers to command a price that exceeds the cost of making additional units. So one unit is transacted at a price between the cost of making and the benefit to the customer. Think here of a suit of armor. Suppose a tall nobleman orders a suit to fit. Making several like it won’t help the maker if the nobleman wants only one,
because a suit made to fit the nobleman is of little value to others of a different bodily shape and size. Costs and thus prices for “manufactured” goods in this world are high. Few people have access to manufactured goods. Transactions happen in small numbers, by modern standards. An economy composed of rare transactions supports a dismal standard of living for most people.

![Figure 6: Transactions resulting from Ancient Making](image)

If we make the added assumption that goods will be manufactured for individuals who will transact for one unit of a given item (that the only transaction that will occur is at \( x=1 \) as depicted in Figure 6), we can simplify our benefit calculation.\(^4\) In this situation, each

\[^4\text{We believe that this assumption is uncontroversial as a description of an ancient making world. Even if a second “copy” of an item could be sold at a profit, the difficulty in ancient contexts, in which communication was slow and difficult, of locating the specific buyer with willingness to pay a price greater than the cost might have been prohibitively difficult.}\]
time an item is produced, it will produce the core and individual benefits \((B_i + B_c)\) and incur the cost of iteration, \(C_i\). Then the maker will go to the next customer and the next item to be made. For a single, unique item produced, then, profit becomes

\[
\delta_p = (B_i + B_c) - C_i
\]  

(8)

and the number of unique items produced, \(y\), will yield a total profit

\[
\delta = \delta_p y
\]  

(9)

If we assume further that the maker has a limited budget, \(C_{\text{MAX}}\), which his costs cannot exceed—think of this as the amount of time the blacksmith has in a day, after which is fire cools and he must recharge his being and supplies, we can determine the maximum profit that can be made using ancient making

\[
\delta_{\text{AN}}^{\text{MAX}} = [(B_i + B_c) - C_i] y_{\text{MAX}}
\]  

(10)

where \(y_{\text{MAX}} = C_{\text{MAX}}/ C_i\); thus

\[
\delta_{\text{AN}}^{\text{MAX}} = (B_i + B_c)( C_{\text{MAX}}/ C_i) - C_{\text{MAX}}
\]  

(11)

The expression in (10), the maximum possible profit that can be achieved using ancient making will be useful for a comparison to the maximum profit available from other ways of structuring work. We turn now to another way of making, "industrial making."

3.2 Industrial Making

When Eli Whitney offered to manufacture 15,000 muskets for the U.S. Treasury, he had established a national reputation as a maker of machines, but he had only recently seen a musket up close (Mirsky and Nevins, 1952). He chose to make muskets after deciding that only the national government had the resources he needed to implement his as-yet-untried “interchangeable system.” The product itself was of no great interest to him. In thinking about how to structure his new factory before he decided what to make in it, he
was thinking in a way that an ancient maker, focused on the unique product and devising a unique process to create it, could not have.

To construct his making system, Whitney analyzed a musket into its parts and designed a machine to make each part. Parts made this way achieved unprecedented levels of accuracy in dimensions and shape, so much so that they were \textit{interchangeable}. A conceptual advantage in thinking Whitney’s way of making was that the design of the product could be abstracted from the process, before any part was made. The product and process for making it were preconceived. Later, Henry Ford demonstrated the dramatic economic advantages of this approach by extending it to the concept of “mass production.” In the spring of 1913, James Purdy, in charge of magneto assembly at Ford’s Highland Park factory, divided the job into 29 standard operations performed by 29 men as the magnetos passed in front of them on a continuously moving belt. This arrangement cut assembly time from about 20 minutes to an average of 13 minutes and 10 seconds. Later refinements dropped the time further to five minutes. This idea began to be used widely in the plant. In August 1913, it took 12.5 hours to assemble a chassis; by January 1914, it took only one hour and 33 minutes (Nevins, 1954). The number of cars produced increased from 78,440 in 1911–12, to 730,041 in 1916–17.

Frederick Taylor applied the ideas Whitney and Ford used for parts and chassis to the workers themselves. He broke jobs down into their smallest gestures and timed every move, suggesting improvements in arrangements and individual gestures, until he could honestly present his rate, method, and quota as the best way to do the best possible job with the least wasted effort. By doing the job exactly according to one of Taylor’s instruction cards, a worker could increase, even double, the day’s pay. By failing to do the job exactly as ordered, a worker could reduce the day’s pay alarmingly. Taylor wanted no individual initiative. He applied Whitney’s idea of interchangeable parts to the workers in the shop; they became units of labor, easily moved from job to job. Control of
making passed from workers skilled in making, to workers skilled in designing the processes for making. With the essential expertise resident in the process designer the firm could hire less skilled, less expensive workers to do the actual work.

The changes that accompanied the industrial revolution changed the structure of work, rendering it more *sequential*. To achieve interchangeability of musket parts, and to create the productivity gains of Ford’s factory, every part of every product had to be specified in advance. Taylor’s process designers similarly pre-specified every aspect of the process of making parts and products. Design activities became separate from production activities. The people responsible for advance specification of parts and processes took up residence in separate areas of the factory and eventually left for separate white-collar buildings.

We can express the logic of industrial advances in terms already introduced this way: If high costs of reconfiguration and exploration limit our ability to transact, *stop incurring these costs in production*. How? *Stop adjusting each thing to its unique purpose; let the customer adjust.*

Like many great ideas, this one is simple, obvious in hindsight, and radically subversive in its time. Industrial methods don’t avoid the costs of reconfiguration and exploration, but only extract them from making itself, and place them at the front end of the process, transforming them into what we now call “product development” and “process engineering” (or, in software development, “requirements definition” and “analysis”). The costs associated with reconfiguration and exploration in these early stages of a sequential process can then be shared by a large number of units emerging from a standardized manufacturing process. Equally important, through application of technology and improvements in process design, variable cost can be reduced, often dramatically. The average cost of making one unit of the made thing is reduced. The important insight was this: *if costs can be lowered enough, customers won’t mind that*
the product no longer perfectly suits them. Many customers will happily buy a knife that, although not perfectly suited to them, cuts pretty well, provided its price is low enough. They will trade unique perfection for affordability and availability.

If we consider industrial making in terms of its benefit and cost characteristics, we note that this way of working does not aspire to provide individual benefits to a particular person, only core benefit. Thus,

$$B_{TOT} = B_C x \quad (12)$$

and

$$B_{AVG} = B_C \quad (13)$$

Cost too has a different structure, since industrial making sequentially separates reconfiguration and exploration cost from variable cost. Reconfiguration and exploration costs are experienced once at the beginning of the process, then variable costs are generated as each identical item is made. Let us use $C_{RE}$ to denote the sum of reconfiguration and exploration costs. Then,

$$C_{TOT} = C_{RE} + C_v x \quad (14)$$

where $x$ is the number of identical knives made, and

$$C_{AVG} = C_{RE}/x + C_v. \quad (15)$$

Notice, that now, in contrast with ancient making, the average benefit from knives is constant and the average cost falls and approaches the variable cost, $C_v$, asymptotically as $x$ increases. The first knife of a kind incurs reconfiguration and exploration cost, as well as variable cost, but each subsequent knife incurs only variable cost. As the number of knives made grows large, the costs of reconfiguration and exploration become less consequential.

Profit from industrial making is

$$\text{IN} = B_C x - (C_{RE}/x + C_v)x \quad (16)$$
There is an uninteresting case in which profit is always less than zero, because variable costs are greater than core benefits, but we set that aside in favor of the more interesting case in which

\[ B_c > C_v \text{ and } C_{RE} + C_v > B_c \]  \hspace{1cm} (17)

Figure 7 depicts this situation graphically. As before, transactions occur only when profit is greater than zero. Note, transactions are possible only when large numbers of like items are made. In this industrial making world, transactions occur when customers purchase generically designed products not tailored specifically for their individual uses. Products have core benefit to individuals that exceeds their rather low cost, provided that we make enough of each product to drop the unit cost to a sufficiently low level. Costs and thus prices for “manufactured” goods in this world are thus low.
Many people have access to manufactured goods. Transactions happen in large numbers, which supports a much-improved standard of living for most people.

The maker again works within a budget, $C_{\text{MAX}}$; thus we can derive the maximum profit achievable using industrial making as

$$?^{\text{IN}}_{\text{MAX}} = B_C x_{\text{MAX}} - (C_{\text{RE}}/x_{\text{MAX}} + C_v)x_{\text{MAX}}$$

where $x_{\text{MAX}} = (C_{\text{MAX}} - C_{\text{RE}})/C_v$; simplifying yields

$$?^{\text{IN}}_{\text{MAX}} = B_C [(C_{\text{MAX}} - C_{\text{RE}})/C_v] - C_{\text{MAX}}$$

(19)

By comparing (11) and (19) as we vary underlying cost components, we can better understand the rationales for structuring productive activities in different ways. Those comparisons are the subject of the next section.

3.3 Comparing Ancient and Industrial Making

We can explore graphically how profit will change in different conditions, and as we structure work in different ways. Begin with conditions facing the ancient maker. The ancient maker, who has access to little productivity enhancing technology, faces high reconfiguration, exploration, and variable costs relative to his budget, and makes a product with a large component of individual benefit. For example, let

$$B_i = 14/15 \cdot C_{\text{MAX}} \quad \text{and} \quad B_C = 1/5 \cdot C_{\text{MAX}} \quad \text{and} \quad C_v = 2/5 \cdot C_{\text{MAX}}$$

Because $C_{\text{RE}}$ will be of particular interest later, we allow it to vary and plot profit against it. Note, however, that if $C_{\text{RE}}$ is of a magnitude similar to $C_v$, and if we assume $C_{\text{MAX}}$ is the ancient maker’s effort budget for a day, then he can make roughly one item (or a little more) per day. So the ancient maker does not produce his product very quickly in these conditions.
Figure 8: Profit comparisons with starting values

Figure 8 shows what happens when we plot the profits from ancient making as described in (11) against the profits of industrial making as described in (19) in these conditions. Only ancient making produces a profit in these conditions, and then only when \( C_{RE} \) takes on sufficiently low values (less than 115 or so). With costs high relative to the maker’s budget, he cannot make enough units to achieve scale economies, so there is no reason in such conditions even to consider arranging work industrially.

---

5 Numerical values used in constructing the example shown in this graph are: \( B_i = 140 \), \( B_C = 30 \), \( C_V = 60 \), and \( C_{MAX} = 150 \).
Imagine, now, however, that new technologies and methods begin to improve along the trajectory that includes Whitney, Ford, and Taylor. More specifically, we begin to focus technology and methods on improving productivity and efficiency by lowering variable costs. If we reduce costs far enough, say to $C_v = 1/30 \times C_{MAX}$, the profit comparison changes as depicted in Figure 9. Even if $C_{RE}$ does not change much at all, the reduction in variable costs makes it profitable to structure work industrially. The reason is intuitive: with variable costs reduced, we have created the potential for economies of scale. A similar (or additional) such effect can be obtained from applying technology to increase the cost budget, $C_{MAX}$. Notice also that profits are up and that more items are made and sold.

Figure 9: Profit comparison when variable cost is reduced
Indeed, the great progress made possible by convergence of ideas and technologies we call the “industrial revolution” was primarily due to reducing variable cost ($C_v$). By comparison, the total reconfiguration and exploration cost ($C_{RE}$) were probably increased due to the additional care and precision required to preconceive in detail the product and the manufacturing process. The increase in reconfiguration and exploration cost mattered little, however, because those costs were extracted from manufacturing and incurred only once at the beginning of making, into the product development and process design stages, thus these costs could be spread over the many like units that would be manufactured. What Figure 9 shows us is that if the conditions of an ancient world are changed by improvements in technology and methods so that variable costs can be drastically reduced, then there are gains from organizing work in an industrial manner.

This maps back to our software development example as well. As the productivity gains from 3GLs and structured methods were introduced, the variable cost of producing lines of computer code was driven down. At the same time, a market for identical or almost identical software packages emerged. The rationale for organizing software production in a sequential, planning intensive manner was thereby strengthened. It remains to be seen how agile approaches emerge in this story.

**3.4 The Emergence of Knowledge Intensive (Agile) Making**

Cost reduction is as important to recent transitions in the nature of work as it was in the transition from ancient to industrial making. The move to knowledge intensive making (of which agile software development is an example), results however from reconfiguration and exploration cost reduction, rather than variable cost reduction. Computing industry
pioneer J. C. R. Licklider understood this in 1960, when he wrote of his dissatisfaction with how computers were used then:

You formulate your problem today. Tomorrow you spend with a programmer. Next week the computer devotes 5 minutes to assembling your program and 47 seconds to calculating the answer to your problem. You get a sheet of paper 20 feet long, full of numbers that, instead of providing a final solution, only suggest a tactic that should be explored by simulation.

The problem, as Licklider expressed it, was that in ordinary intellectual work, whether assisted by computers or not, “About 85% of my ‘thinking’ time was spent getting into a position to think, to make decisions, to learn something I needed to know.” The prescient Licklider realized that the computer could reduce that 85% and allow humans to spend more time on formulation, insight, learning, and improvement. To do that, though, computers had to become much easier to reconfigure. The problem was not variable cost (the computer spent only 47 seconds calculating), but with front-end, setup costs.

Licklider understood too that lowering the cost of reconfiguration would permit problem solving to be structured in a more useful (iterative) way:

Present-day [i.e., circa 1960] computers are designed primarily to solve preformulated problems or to process data according to predetermined procedures. The course of the computation may be conditional upon results obtained during the computation, but all the alternatives must be foreseen in advance…However, many problems that can be thought through in advance are very difficult to think through in advance. They would be easier to solve, and they could be solved faster, through an intuitively guided trial-and-error procedure in which the computer cooperated, turning up flaws in the reasoning or revealing unexpected turns in the solution (p. 6).

Similar logic could be applied to ways of making. The great benefits of the industrial revolution resulted from the efficiencies available when “preformulated” products were built according to “predetermined procedures.” Of course different products could be produced by industrial systems, but “the alternatives [had to be] foreseen in advance.” In creative activities that involve discovery (such as problem solving, or creation of new products, e.g., software), however, it would be easier and go faster if we could use an
“intuitively guided trial-and-error procedure in which the computer cooperated, turning up flaws in the reasoning or revealing unexpected turns.” Licklider is writing here about what we have called “exploration costs” as well (not just reconfiguration costs). If the trial-and-error that Licklider proposed incurred massive costs of any kind, this would present a problem. And just as application of technology reduced variable cost, the application of technology can reduce reconfiguration and exploration costs. Thomke (2003) has outlined numerous ways that technology is applied to reduce cost of “experimentation” in a variety of contexts, such as product development and drug discovery. Examples of reconfiguration and exploration cost reduction technologies include: simulation (of, say, crash tests in automaking) software, robotic experimentation equipment, version control systems (which permit rolling back from “mistakes” at low cost), and rapid prototype generation technologies (such as daily build setups in software development).

Figure 10 shows what happens if we start from the situation depicted in Figure 9 and then reduce $C_{RE}$; note that the profit lines eventually re-cross (in the circled area in Figure 10) and the highly customized approach we called ancient making earlier re-emerges as the recommended approach. Because these conditions are very different than those that justified ancient making, and because the nature of the work, though similar in structure, is different in many other ways from that of the ancient craftsman, we choose to call this re-emergent form of making by a different name: “knowledge intensive making” or “artful making.”

The re-crossing of these lines has a major effect on the recommended work structure. At the point where the lines re-cross, the rationale for sequential structure collapses and the industrial revolution tactic of manufacturing for core benefit alone is rightfully abandoned. As with ancient making, the making process again co-mingles

---

6 For more information about “artful making,” see Austin and Devin (2003).
design and making rather than separating them into separate stages. Makers begin customizing for particular customers. Production stops being just about execution and begins to be about exploration and discovery, in every iteration.

![Figure 10: How Reducing $C_{RE}$ Changes Work Structure](image)

4. **Using Information Technology to Support Making**

The simple model outlined above has implications that help us understand how work evolves and how it should be structured. We suggest that the application of technology, and especially information technology, follows a historical pattern.
1. Beginning state (Ancient Making): Low productivity, work as craft, iterative structure that co-mingles exploration and production, singular and unique outputs → Lousy economics (small numbers of transactions)

2. Technology is applied to reduce variable cost (often raising reconfiguration and exploration cost), creating the potential for industrialization

3. Interim state (Industrial Making): High productivity, work as standardized action, sequential structure that separates exploration and production, standardized outputs → Vastly improved economics (large number of transactions)

4. With variable cost low, technology is applied to reduce reconfiguration and exploration costs, creating the potential for knowledge intensive work.

5. End State (Artful Making): High productivity, work as ongoing innovation, iterative structure that co-mingles exploration and production, singular and unique outputs → Improved economics (even more transactions and value capture, as customized products capture value in niches not accessible to standardized products.

Figure 11 shows graphically this historical pattern.

It is interesting to consider why technology is first applied to reduce variable cost, then later reconfiguration and exploration costs. Empirical evidence that it does appears strong. Henry Ford’s reductions in the time it took to assemble a magneto or a chassis clearly predate by decades Computer Aided Design and other technologies that reduce the costs of preconceiving products and processes. We suggest that reducing variable cost is simpler because it requires application of technology to repetitive actions. Reducing reconfiguration and exploration costs, by contrast, requires that technology support non-repetitive actions. And it is easier to design technology to do the same thing again and again very efficiently, than it is to design technology to support novel activities.
The inherent costs of reconfiguration and exploration in a making process are related to the *pliability* of the materials used. If we are working with metal, a material that is relatively un-pliable, it is simple to improve the efficiency of grinding by using a mechanical wheel, but it is more difficult to reduce the costs of reconfiguration and exploration, because the heavily-built tools that bend or grind metal tend to be difficult to change so that they do a good job in a novel way (reconfiguration cost), and because metal is difficult to unbend or ungrind (exploration cost). If we are dealing with a more pliable material, say wet clay, then costs of reconfiguration and exploration may be more easily attacked (though even then if there is kiln firing at the end, this may be the source of high costs). In terms of inherent pliability, however, the highest pliability for a working material is achieved though *virtualization*—i.e., rendering the material and its properties
into bits and bytes, which can be rearranged at very low cost. We suggest that only by applying information technology to virtualize work—using simulation, automated experimentation, rapid prototype generation, etc.—can most kinds of physical work re-cross the line from industrial to artful process structure.

It is here, however, that we can return to our concern about the structure of software development. The materials of this kind of work (lines of code in a file) are already virtual. Thus it is quite natural that software development would move rapidly through the described historical pattern and emerge quickly into agile process structures. Even though software is already virtual, it does require some supporting technologies to reduce reconfiguration and exploration costs, such as daily build technologies, version control systems, and automated test suites. We suggest that agile approaches arise at this juncture because the support technologies are becoming good enough to support low cost configuration and exploration. Fifteen years ago, daily builds and automated testing were harder to achieve, thus reconfiguration and exploration costs were relatively high, and the case for industrial making of software was much stronger.

By this logic, as more physical work is successfully virtualized, its structure will become much more like agile software development. Rather than software development converting to be more like “real engineering and manufacturing process,” we may instead see engineering and manufacturing processes become more like agile development, as materials associated with this kind of work are increasingly virtualized.

4.1 Demand for Innovation

So far we have been mostly discussing costs. There is another factor that matters to process structure, however: benefit. Specifically, the relative magnitudes of the two components of benefit, $B_i$ and $B_c$, express the demand for innovation in a particular product category. Customization matters more in some product categories than in
others. For example, knives, which allow anyone to cut meat, probably offer relatively low individual benefit relative to, say, suits of armor, which might chafe badly if they are not a good fit. For some products, like knives, the majority of their benefit comes from the core component of benefit; for others, like suits of armor, the majority of benefit comes from the individual component of benefit. Whether this is true depends also on the context of use; for knives intended not to cut meat, but rather to allow a knife thrower to compete successfully in a world championship knife-throwing event, the individual benefit component might far outweigh the core benefit component. It is safe to say, though, that as $B_i$ increases relative to $B_c$, the demand for innovation increases.

![Figure 12: When Demand for Innovation Approaches Zero](image)
As we might expect, when $B_i$ increases the attractiveness of industrialization decreases. Conversely, as $B_i$ approaches zero, the rational for using anything other than industrial making disappears. This effect is intuitive; as the demand for uniqueness in a product decreases, the advantages of using a process designed to produce unique product features disappears. Figure 12 shows what happens when $B_i$ approaches zero.

4.2 Predictions

This model, though simple, allows us to make a number of predictions that could be addressed in future empirical work. The model suggests that the primary factors in determining works structure are 1) variable cost, 2) reconfiguration and exploration costs, and 3) demand for innovation, as summarized in model parameters.

Prediction 1: When variable, reconfiguration, and exploration costs are high, and demand for innovation is high, ancient making (work as craft, iterative structure, low productivity) should prevail.

Prediction 2: When variable costs are low, reconfiguration and exploration costs are high, and demand for innovation is not great, industrial making (standardized work, sequential structure, high productivity) should prevail.

Prediction 3: When variable costs are low, and reconfiguration and exploration costs are low, artful making (work as ongoing innovation, iterative structure, high productivity) should prevail.

Prediction 4: If demand for innovation is high enough, industrial making will not arise.

Prediction 5: If demand for innovation is near zero, then only industrial making will arise.

Prediction 6: Work in which material pliability is high holds great potential for artful making and will tend to transition toward that approach to structuring work.

Prediction 7: Work will tend toward artful making as it is successfully virtualized via the application of information technology.
4.3 Practical Implications

The model also suggests some practical implications about the future use of information technology. One could argue that the past (highly successful) history of applying IT to business has resulted primarily in variable cost reductions, thus has been part of the transition from ancient to industrial making. Indeed, even today IT systems are usually financially justified by increases in efficiency and commensurate reductions in cost. As we reach the limits on what we can achieve by applying IT in this way, we should see a shift to using IT to reduce reconfiguration and exploration costs. This implies that future gains from application of IT will not result as much from efficiency gain and cost reduction as from the facilitation of innovation and revenue enhancement. We have already mentioned simulation, computer aided design, rapid prototype generation, and automated experimentation as areas in which IT is being applied in exactly such a manner. It is also worth noting that communicative uses of IT, to support video teleconferencing and other collaborative activities, is also arguably an example of this kind of IT application.

Spending on IT to reduce reconfiguration and exploration cost is of a different form that the traditional spending on variable cost reduction. Anecdotal evidence suggests that we do not yet sufficiently understand this distinction. It is not hard to find examples of managers asking for project justification based on variable cost reduction for projects that are primarily intended to reduce reconfiguration and exploration costs. Seeing the benefits of reduced reconfiguration and exploration costs remains a challenge to some business imaginations.

5. Precedents: Processes in the Arts

This research project began with the observation that agile software development and collaborative arts rehearsal processes have a surprisingly similar process structure. The
model helps us understand why this is. While the two activities seem to abide in far distant realms, their underlying cost structures and demand for innovation are quite similar. The actions of actors, like the bytes that make up computer code, are a pliable material, with potentially low reconfiguration, exploration, and variable costs. The product of a theatre company, a play, is made up human actions that are naturally suited to iterative structure. The script supplies some “requirements,” but the excellence of the product depends critically on ideas and innovations gathered along the way.

Computer code requires a little more technology to enable iterative structure, such as rapid build technologies and version control system. But in the presence of these technology supports, the two activities look remarkably alike. Both proceed in cycles of trying things, discussing the results, and then trying something else. In business terms, rehearsal in a theatre company is a rapid prototyping process. One might also observe that in a good theatre company, the prototyping process is very successful at producing innovation—new and interesting products—on a rather severe deadline—opening night. Few businesses operate under deadlines so severe; fewer still innovate successfully and reliably under such deadlines.

Consider this incident as described in our notes from field research at the People’s Light and Theatre Company, when we were attending a performance of Athol Fugard’s play *The Road to Mecca*:

I sat in just the right spot to see this. One of the characters was setting a table—plates, cutlery, and glasses with stems—while conversing with another character. At a tense moment in the conversation, the actor setting the table accidentally bumped a glass with the back of her hand, causing it to fall onto the seat of a chair, on its way to the floor. Now

---

A common misconception among those not involved with the theatre is that the script is the equivalent of a “specification” in an industrial process. Brief consideration of this idea reveals that it is problematic, however. A play is not nearly detailed enough to serve as a specification. Further, the excellence of a play depends on much more than executing the script exactly as specified. Indeed, a play that achieved nothing more than getting the words exactly right and following the stage directions in the script would likely be considered a lifeless and unoriginal creation.
obviously this could have been very disruptive. Broken glass all over the stage? But she incorporated it into the performance. The chair had a cushion, so the glass bounced. She caught the glass mid-bounce and then froze her gaze on the other character, as if the whole thing were a reaction to what had just been said. No way had she rehearsed knocking the glass off the table and catching it on the bounce; but when it happened, she made it a part of her work.

Although the dropped glass was not a part of the play on subsequent nights, the remnants of the event—the forceful dialogue echoed in sharp hand movements, a pause holding the glass at just that moment—were apparent in future performance, and added to the overall effectiveness of the scene.

This episode illustrates an advantage that artful processes have over industrial processes. If we were talking in software development terms, the falling glass might represent a new system requirement that had appeared unexpectedly and suddenly. An artful process, which co-mingles design and production, and that presumes unique elements in the work each time, is well prepared for such a late breaking requirement. Artful processes are adaptable. But thinking of the falling glass as simply a new requirement, something new to be dealt with that is not central to the main purpose of the work, underestimates the strengths of artful process. Better to think of the falling glass as an opportunity for found innovation; and to note that this process is structured iteratively precisely for the purpose of consistently and reliably finding innovation.

Interestingly, arts processes may provide some of the best exemplars of processes in which serendipitous insight plays an important role. Much has been made recently of the apparent failure despite invested billions of “industrialized” drug research (Sherman and Ross, 2003), which aims to accelerate drug discovery by speeding up experimentation. Critics charge that the technologies do not present opportunities for found innovation. A recent Wall Street Journal article (Landers, 2004) offers the example of an experiment that failed to produce the intended result, but that had a fascinating and important side effect that a human experimenter noticed and that lead to the discovery of
a billion dollar drug; a robot would have simply chocked up the result as a failure. Indeed, Fleming’s famous discovery of penicillin was just such a serendipitous discovery: He noticed that bacteria accidentally left growing in a dish would not grow near a spot of mold.

While noting the advantages of artful process, however, we should note as well that even some arts activities are not (and should not be) structured as artful processes. A commercial production of a much loved musical may, in fact, demand little innovation (\(B\) may be low, because people want to see it just as they remembered it from when they saw it before), and therefore have little reason to employ an artful process. Examination of a commercial performance of say, *Cats*, would reveal a much more sequential and industrial process than the one that produced Fugard’s *Road to Mecca* at the People’s Light.\(^8\) Even productions at the People’s Light have industrial elements, such as design of physical sets, which often have lead times and must be arranged sequentially. Abigail Adams, artistic director at the People’s Light, told us that a fundamental challenge for artful managers is to manage the co-evolution of artful and industrial elements so that they combine to good effect. A similar challenge may face makers of software, strategy, new products, and other innovative outputs in business.

5.1 *Reconceiving versus Replicating*

Repetition is a fundamental characteristic of productive work. Much of what we do when we create value must be done again and again. Information technologies have been particularly useful in supporting the repetition required when we take useful actions. “Automating” a manufacturing process, for example, typically means applying technology to make repeated processes work more consistently and efficiently. When people

---

\(^8\) We are indebted to Kevin Hendricks, of the Richard Ivey School of Business at the University of Western Ontario, for suggesting this example.
interactively explore to try to make something new or solve a novel problem, they also employ repetition, using trial-and-error or experimentation to learn more about the problem and its possible solutions. You might not repeat exactly the same motions as you do in a manufacturing plant, but you are still usually trying something again and again, iterating through a process that is similar from repetition to repetition. Thinking about these different varieties of repetition, it is helpful to distinguish between reconceiving, the fundamental act in artful making, and replicating, the fundamental act in industrial making (Austin and Devin, 2003). We deal first with replicating because it is more familiar in technological contexts.

Replicating aspires to outcomes that conform as exactly as possible to an objective or specification defined in advance. An automobile assembly line, for instance, might replicate Ford Taurus LXs with standard options in spruce green clear coat metallic. The company intends to make all such vehicles identical (even if they intend to produce them only every fifth car for two days each month). What constitutes a good car and a good process for making a car are both known in detail in advance. Technology helps the automakers be sure that what they actually make conforms to what they conceived in advance. Or, consider the Interactive Voice Response (IVR) system that you hear when you call your bank: It replicates the service of providing balance information. The bank intends to serve every balance inquiry in exactly the same way (except for actual dollar amounts retrieved at the end). The technology assures this. If the outcomes of replicating are not identical when they are intended to be, there is a name for this eventuality: a “quality problem.” Much of the conceptual and physical machinery of industrial production focuses on perfecting replication, to enhance efficiency and avoid quality problems.

Reconceiving, in contrast, never tries to produce an outcome precisely defined in advance. People reconceiving may loop repeatedly through stages of—say—
brainstorming, problem definition/reformulation, experimentation and analysis, evaluation, and discussion. But this repetition does not seek conformity to preconceptions of outcome or process. Often reconceiving seeks the opposite: freedom from the preconceptions and other constraints on creative thinking. Reconceiving takes conflicting circumstances, materials, and outcomes, and makes from them a new set of circumstances, materials, and outcomes. The new circumstances, materials, and outcomes tend to be more valuable than the preceding ones, but not every time. Some repetitions “regress,” producing outcomes that don’t work as well as earlier ones. Progress tends to be incremental. Now and then reconceiving yields innovative leaps that surprise everyone. Reconceiving is the approach we usually use when we seek a new solution to a problem, a new product or service, or a new point of view on an old situation.

Table 1 identifies differences between replicating and reconceiving as they are used in a variety of activities. In reconceiving, one often “discovers” an outcome—a strategy or a plan, say—at the same time that he or she is creating it. The action involved in reconceiving may converge to some degree of regularity in outcomes—subsequent strategies or designs from a reconceiving process may resemble each other—but the action never converges completely, nor do those involved in this kind of activity aspire to complete convergence. Because reconceiving does not involve prespecification of an objective, it is capable of an infinite number of outcomes in subtle variations, rather than a finite number of prespecified ones. And there is a further difference: Replication processes are often managed to protect them against uncertainty in inputs, demand, and environmental conditions; whereas reconception processes instead treat uncertainty as fuel for adaptation and innovation. Such a process, which is not constrained by a specification, and which leaves itself open to environmental variation, retains a significant ability to adapt, to make anew. When people are
reconceiving, production shifts from trying to “get it right the first time” to instead trying to “making it great before the deadline,” which more accurately describes what is needed in many collaboration situations.

<table>
<thead>
<tr>
<th>Replicating</th>
<th>Reconceiving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Produces pre-specified outcomes</td>
<td>Produces outcomes that were not pre-specified</td>
</tr>
<tr>
<td>Design and manufacturing are separate, sequential phases; the process for producing the pre-specified desired outcome is also designed in advance</td>
<td>Design and manufacturing are part of each other; the process that produces a desired outcome is the same process by which the desired outcome is discovered</td>
</tr>
<tr>
<td>Produces a finite and discrete number of outcomes (e.g., a car in red, green, or blue)</td>
<td>Can produce an infinite number of unique outcomes, including “in-between” outcomes (e.g., a car that is “a little darker green than that”)</td>
</tr>
<tr>
<td>Making processes are buffered against variation</td>
<td>Making processes seek out and are driven by variation</td>
</tr>
<tr>
<td>Well-defined notion of final outcome</td>
<td>Ongoing adjustment of outcome; notion of a “final product” problematic</td>
</tr>
</tbody>
</table>

Table 1: Replicating versus Reconceiving

This more nuanced view of repetition has important implications for how technology should be deployed to support work processes. Technology should be used very differently in support of reconceiving than in support of replicating. It is necessary to emphasize this point because the use of information technology to support replicating has been so widespread and successful. When people think of applying information technology to work processes, they often think of end-to-end automation, which imposes uniformity on the way work can be accomplished. Indeed, often this uniformity is equated with quality and is the very point of automation. But this is not the right approach when we are seeking to produce rich, real-time collaborative technologies.

Technologies to help in reconceiving must support trying things again and again, but in different ways each time. The objective is not to enforce uniformity, but to support
ongoing innovation and change. Whereas information technology has often been successfully used to reduce costs and improve quality by standardizing, streamlining, and reducing variation, the technology needed to support collaboration will often be aimed at creating something new with each try. Such a process may yield value more often in revenue enhancement—creation of new products, expansion of markets, development of new strategies—than in cost reduction. To support reconceiving, technologies will need to be flexible, and rapidly and cheaply reconfigurable, because they will need to support tasks that cannot be known in advance, that must be discovered during the process. We believe reconceiving is the basis of a different form of work, which should be managed differently, and to which technology should be differently applied. The specifics of this approach to management we have detailed elsewhere (Austin and Devin, 2003).

Conclusion

In this paper, we have proposed a simple model to relate (1) the demand for novelty in a product and (2) the underlying cost structure of production tasks, to (3) the way in which development should be structured. In devising this model, we sought to address questions implicit in the “methodology debate” (Highsmith, 2001) between developers who emphasize planning before coding and those who emphasize going early to code prototypes. The model specifies the conditions in which developers should use each approach, and suggests that both sides in the methodology debates may be correct, depending on the characteristics of the environments in which they work. We frame our arguments within a discussion of how software development and other productive activities have changed historically, and intend to derive a simple theory that applies to productive activities beyond software development. Framed in historical context, the model suggests that the emergence of “agile” software development might exemplify a
more general transformation in the nature of work. We note similarities among agile
development, evolving approaches to other kinds of knowledge work, and long-
established practices in the collaborative arts (e.g., theatre), based on similarities in
circumstances and underlying cost structures. The model offers falsifiable empirical
predictions that can be tested in future research. It also produces practical advice that
may be useful in thinking about investments in information technology.
References


________, remarks made during a presentation at the Cutter Summit, April 29–May 1, 2001, Cambridge, MA.


