SOFTWARE PRODUCTION ECONOMICS: THEORETICAL MODELS AND PRACTICAL TOOLS

Chris F. Kemerer

April 1988

CISR WP No. 168
Sloan WP No. 2005-88

Center for Information Systems Research

Massachusetts Institute of Technology
Sloan School of Management
77 Massachusetts Avenue
Cambridge, Massachusetts, 02139

SOFTWARE PRODUCTION ECONOMICS: THEORETICAL MODELS AND PRACTICAL TOOLS

Chris F. Kemerer

April 1988

CISR WP No. 168
Sloan WP No. 2005-88

To be presented at the ACM 27th Annual Technical Symposium, Washington, D.C., June 1988.

Center for Information Systems Research
Sloan School of Management
Massachusetts Institute of Technology

SOFTWARE PRODUCTION ECONOMICS: THEORETICAL MODELS AND PRACTICAL TOOLS

CHRIS F. KEMERER

MIT Sloan School of Management
E53-329, 50 Memorial Drive
Cambridge, MA 02139
617/253-2971
BITNET: CKEMERER@SLOAN

To be presented at the ACM 27th Annual Technical Symposium, Washington, D.C., June 1988.

ABSTRACT

If the current work in software engineering is to have its maximum impact, more attention needs to be focused on methodological tools for understanding and managing the software development process. This paper proposes the use of microeconomics to model software development as a production process. Microeconomics provides both a rich theory with which to model the likely impacts of software tools and a diverse set of analytic tools with which to validate those models on real world data. One theoretical model, Galbraith's Technology Imperative Model, is described and then applied to the software development process to illustrate the insights available from a microeconomic approach. The hypotheses proposed from such a model can be empirically tested, and current examples of such research are presented. Finally, a substantial list of additional research questions stemming from the microeconomic production process model are presented, with suggestions on how to continue the work begun in this area. The benefits from such research include not only a greater intellectual understanding of the software development process, but also some immediately valuable recommendations to practicing software development managers.

ACM CR Categories and Subject Descriptors: D.2.2 [SOFTWARE ENGINEERING] Tools and techniques, D.2.8 [SOFTWARE ENGINEERING] Metrics, D.2.9 [SOFTWARE ENGINEERING] Management K.6.0 [MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS] General, K.6.3 [MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS] Software Management.

I. INTRODUCTION

In order to effect the move from a labor-intensive "craft environment" to a tool, or capital-intensive "production environment" it will be important to develop models of how software is developed under a production approach. A number of such models have been proposed. These models typically either deal only with one aspect of software development (e.g., design [Spector/ Gifford 86]), or address only certain types of software development (e.g., programming-in-the-small.)[1].

What are needed are broader, more general models of how to apply engineering concepts to the task of software production. One such model is the "software factory" concept [Bratman/Court 75], whose popularity has waxed and waned, but may be on the rise again due in part to some reported successes in Japan [Cusumano 87]. This concept draws a very specific analogy between factories, which are strictly manufacturing entities, and software production. Software, particularly custom software, while typically thought of as a product in the software factory model, has many characteristics of a service. These include the greater customer participation in the creation of a service, the less tangible nature of services, the greater difficulty in measuring the output of a service, the lesser degree of standardization of a service, and the tendency of service operations to locate closely to their customers [Walls 86]. For these reasons, more general models of software production would also be useful.

Fortunately, such models are available. Economists define "production" as any activity that creates value. In addition, they have developed a number of powerful analytic tools with which to quantitatively model any production process that transforms inputs into outputs. The proper application of these models and analytic tools can have several obvious benefits for the software engineering community, including directing tool-making efforts where they might find their

[1]See [Manley 85] for a criticism of the limitations of this approach.

greatest leverage, proposing worthwhile research questions and generally guiding managerial thinking about the software development process. While the process of model-building and the use of models has generally been promoted as an aid to intellectual advancement in many disciplines, software engineering could particularly benefit, given its relatively short tenure and the fact that it exists in a rapidly changing technical and economic environment. The use of models would force the surfacing of assumptions, and thereby possibly expose newly obsolete thinking.

The general outline of this paper is as follows. Section II describes one classic theoretical economic model developed by Galbraith to describe the impact of the increased use of technology on production processes. While this model was originally applied to manufacturing, it is sufficiently powerful to be of use in describing other production applications. Section III discusses its applicability to software production, and uses it to make predictions about what the future may hold for software engineering methods and tools. Section IV then summarizes the results of some actual current research that models software development as an economic production process. These results illustrate the usefulness not only of the economic concepts, but also of the tools of economic analysis. Section V then describes some ongoing and planned research which takes further advantage of the economic models. Concluding remarks are presented in Section VI.

II. Galbraith's Model of Technological Impact on Production Processes

A number of economists have examined the phenomenon of the impacts of technology. Perhaps the best known and the most accessible of these efforts is John Kenneth Galbraith's 1967 book the New Industrial State, now in its fourth edition [Galbraith 85]. In Chapter 2 he outlines what he calls the "imperatives of technology". While much of the rest of the book deals with Galbraith's proposed solutions to his predicted changes, Chapter 2 is limited to describing what Galbraith sees as the effects of technological change. He defines technology as the "systematic application of scientific or other organized knowledge to practical tasks". The use of technology requires that 1) the production task be divided into sub-tasks, 2) that knowledge be applied (typically by specialists) to the sub-tasks, and finally 3) that the finished elements be combined to form the final product or service. While Galbraith has in mind a manufacturing process, (indeed, all of his illustrative examples are from the automobile industry, particularly the Ford Motor Company) it is clear that this general description could apply to any task where technology is being applied. In particular, his description of the change from how Model T's were made -- from highly skilled general labor creating products based on relatively simple systems, to today's decentralized, assembly line production of complex, highly interdependent automobiles -- bears quite a strong resemblance to what the software engineering community is proposing as changes in the development of software. Given this analogy, and his definition of technology that encompasses both methodologies and more conventional tools, it seems reasonable to look at what Galbraith's model of technological change predicts for the future of technologically-aided software development.

Galbraith presents six hypotheses about the effects of increasing the amount of technology used in an economic production processes. The first is that an *increasing span of time* will separate the beginning of the production process, the planning, from its eventual result, the product or service. His example is the difference between the use of readily available sheetmetal for the Model T, and today's complex interconnection between manufacturers such as Ford and their large industrial suppliers. These suppliers may even be involved in the specifications of the raw material (such as steel or tires) that will eventually go into the car. This greater complexity and the need to subdivide the production into smaller tasks has the eventual affect of lengthening the critical path for the entire process.

A second hypothesis is that there will be an *increase in the amount of capital* that is committed to production over and above that merely required for increased output. Put another way, the capital to labor ratio will increase. One reason for this is that the increased knowledge brought to bear on the sub-tasks will eventually be embodied in some tool or methodology. This, plus his earlier assertion, leads to a third hypothesis, that due to the increase in the amount of time and capital that are committed, the *investment in the production process will become increasingly inflexible.* Certainly the massive retooling required by the

American auto makers to meet the demand for smaller cars is adequate testimony to this assertion.

A fourth hypothesis is that increased technology will require *specialized manpower*. As the process of building a car is divided into smaller and smaller tasks, and more and more knowledge (in the form of technology) is applied to those tasks, the staffing requirements change. While an early Ford engineer may have understood how to build an entire car, the current staff members are highly decentralized, each responsible for increasingly smaller parts of the system. Given this specialization, the fifth hypothesis follows immediately. That is that *larger organizations will be required* in order to pull together these disparate workers.

The sixth and last hypothesis follows from the increased time/money commitment, the greater inflexibility of that commitment, and the needs of a larger organization. The sixth hypothesis is that *much greater resources will have to be expended in the planning stage at the beginning of the production process.* Galbraith illustrates this by discussing the relative ease with which Henry Ford could make changes on short notice to his product (leaving aside the question of whether he would do such a thing) compared to the tremendous amount of planning that is required by today's cars, where even the obsolescence is planned.

It will be convenient to summarize these six hypotheses in a short list to make references to them in later sections. They are:

Technology Effects Hypotheses

1. *Increased timespan from inception to implementation*
2. *Increased capitalization (tools)*
3. *Greater inflexibility of the investment*
4. *Greater specialization of labor*
5. *Larger organizations*
6. *Increased planning.*

III. Application of the Technology Model to Software Production

The technology effects described above can be used to describe and make predictions about software development when modeled as an economic production process. In addition to noting whether or not the changes predicted by Galbraith have taken (or will take) place, it is even more important to try and assess the likely *impacts* of such changes. The evidence for these changes and for some of their impacts will be described below and in the research results presented in Section IV.

Galbraith's first (increased timespan) and fifth (larger organizations) hypotheses can be thought of together as an increase in the scale size of production. The question for software production is whether longer projects and increased team sizes are to be expected from the increased use of technology. A number of arguments could be presented for why this is already happening. The development of methodological tools, such as Gane/Sarson/Constantine's structured analysis and design and structured programming have effected exactly the kind of change Galbraith describes, in terms of breaking the production of software into smaller tasks [DeMarco 78]. This modularization is believed to allow for the successful completion of larger projects, as individual team members can work independently and in parallel on different portions of the project that will ultimately come together into a large system. In this regard, the technology, in the form of structured methodologies, may permit the completion of larger projects. Two explanations are possible for explaining the potential creation of larger projects. The first is that project management may choose to tackle more ambitious projects due to the availability of these methodological tools. Alternatively, the fixed costs (startup costs such as learning and in some cases the purchase price of proprietary methodologies) may be sufficiently large that smaller projects may be economically less sound under these methodologies.

A more critical issue, particularly for practitioners, concerns the impacts of larger project sizes. Using the standard definitions of scale economies (i.e., increasing returns to scale are present when the marginal returns to an additional unit of input are greater than the average returns (at a given level of volume), and decreasing returns are present when the opposite is true), does software development exhibit either increasing or decreasing returns to scale? Some recent empirical research by Banker and Kemerer [Banker/Kemerer 87a] related to this question is described in Section IV.

The truth of Galbraith's second hypothesis (increased capitalization/tool usage) is already apparent. Even the brief history of software development can be told through the context of increased tool usage and the resultant increases in capitalization. For example, Jensen [Jensen] categorizes software generations as "Primitive Tools" (e.g., assemblers, basic linkers), "Basic Tools" (e.g., high level language compilers, basic source editors), "Interactive Tools" (e.g., database management systems, interactive debug aids), "Modern Tools" (e.g., virtual memory operating systems, static source analyzers), and "Advanced Tools" (e.g., automated requirement specification languages and analyzers, automated verification systems).

An interesting question to ask in relation to this second hypothesis is, what is the impact on productivity of these tools? Presumably the direction is positive, once the fixed costs such as purchase and learning are subtracted, but how large is the actual impact? Is the effect a difference of degree, whereby productivity on a project is X% higher due to the use of some tool, T, or is it a difference in kind, where certain types or sizes of projects could not be done without the tool? And how can these effects be measured? All of these are important research questions. Some results relating to these questions from a recent study are provided in Section IV.

The third hypothesis, increasing inflexibility, is a likely possibility. Three explanations may be given as to the sources of this inflexibility. The first is the existence of large learning curves associated with learning new methods [Grammas/Klein 85]. Given that a large up front investment is required in new methodologies or tools, firms may be reluctant to adopt them, particularly given the lack of well-documented methods for verifying the payback in the investment. The adoption of Ada (TM, US DoD) may be a case in point [Riddle, cited in

Bayer/Melone 87]. Second, the increasing inventory of software has created both a software maintenance burden and a realization on the part of managers of the large investments that have been made by their firms in software [Parikh 85]. Given the need for staff trained in, for example, COBOL for maintenance work, there may be some incentive for managers to continue to write new software in that language. Third, and finally, as the field matures there will be an emergence of certain de facto standards. Economists have noted that the establishment of standards and the creation of an installed base leads to "excess inertia" in the market, due to the fact that early adopters bear a disproportionate share of the transient incompatibility costs [Farrell/Saloner 86].[2]

The fourth hypothesis, increased specialization of manpower, is related to the previous hypothesis, and there is apparent evidence for it in software production. Early programmers can be likened to Henry Ford's Model T assemblers. They were very close to the machine, were skilled at doing a number of tasks by hand, and generally had a clear understanding of all the steps involved in producing the final product. Today, programming assignments have been abstracted away from the machine level through tools such as high level languages and user-friendly operating systems. Staff today who can, for example, program in assembly language, tend to be specialists. Application programmers have been increasingly supplied with tools that allow them to know less and less about the hardware and systems software with which they are working.

The sixth and final hypothesis, increased resources devoted to the planning activities associated with production, seems to be coming true. As software engineering has matured, increased attention has been paid to methodologies for management, who are the economic actors entrusted with the planning responsibilities. Papers and panels at recent sessions of the International

[2]Farrell and Saloner describe this phenomenon with the memorable phrase, "the penguin effect", since penguins who must enter the water to find food would prefer that another penguin go first, in the event that there are predators. Once the first penguins jump in and appear safe, the rest are glad to follow.

Conference on Software Engineering have included many examples of these methodologies, including models to support managerial planning [Manley 85, Schwartz 87] and algorithmic cost estimation tools [Rubin 85, Miyazaki/Mori 85]. In addition, studies in more managerially-oriented sources continually place planning as the top-ranked issue facing information systems executives [Brancheau/Wetherbe 87].

Again, the issue for software engineering researchers and practitioners is the likely impact of this change. One possibility may be that increased attention to planning may result in more managerially-acceptable projects (e.g., more projects completed on time, within budget, etc.). Given the relatively dismal track record of large systems development projects in these areas (e.g., less than 1% of large completed systems are finished on time, within budget and having met all user requirements, and up to 25% of large projects are cancelled before finishing [Jones 86]) improvement would be highly welcome. However, any improvement via improved planning methods may be hard to identify, given that the methods may induce managers to attempt even more ambitious projects. Methods and metrics are needed that allow comparison of similarly-sized efforts if legitimate comparisons are to be made in order to evaluate the impact of these improved planning tools. Section V describes some research in progress designed to look at one aspect of this process.

In summary, Galbraith's six hypotheses provide a number of insights into how the production of software may be affected by the implementation of technological advances in the form of software engineering tools. This is a theoretical contribution of economics to the understanding of software development, modeled as a microeconomic production process. In the next section, the results of some current research using economic analysis tools to investigate the hypotheses proposed by Galbraith and others are provided.

IV. Current Research Results

Many of the technology hypotheses can be tested empirically, using tools of production economics analysis. In addition, and perhaps of greater interest, the impacts of the changes that are the likely consequences of the increased application of software engineering technology can also be measured. In this section, two recent research efforts will be summarized, and their results outlined. These two examples should serve to illustrate the applicability of microeconomic production analysis to the modeling of software development.

Galbraith's first and fifth hypotheses relate to the increased scale size that is the likely result of the increased application of technology. An issue of importance to both researchers and practitioners is the effect of such an increase in scale size on the production process. Does software production exhibit decreasing returns to scale, so that larger projects are less efficient, increasing returns to scale, so that larger projects are more efficient, or constant returns to scale, where efficiency is not affected by the size of the project?

Plausible theories exist on either side of this question. Researchers such as Boehm [Boehm 81] have noted the presence of a number of factors in new software development that may contribute to increasing returns to scale, particularly software development tools such as on-line debuggers or code generators. These tools may increase productivity, but the relatively large initial investment, both in purchase and in the organizational learning cost, may proscribe their use on small projects. A second factor is that larger projects may also benefit from specialized personnel, whose expertise in a certain area (e.g., assembly language coding) may increase the project's overall productivity. Finally, all projects require a certain fixed investment in project management overhead. This type of overhead (e.g., status meetings and reports) does not increase directly with project size and therefore can be a source of increasing returns to scale for larger projects.

In contrast to this view, many authors have pointed out the possibility of decreasing returns to scale on large software projects. Brooks [Brooks 75] has noted that the number of communication paths between project team members increases geometrically with the number of team members. This communication overhead is a clear case of non-linear cost increase, and hence a factor that could contribute to decreasing returns to scale. Somewhat analogously, Conte, et al. [Conte et al. 86] suggest that larger systems development projects will face more complex interface problems between system components.

Some recent research has set out to empirically test these two competing notions of the returns to scale for new software development [Banker/Kemerer 87a]. Eight published empirical data-sets were tested for the presence of scale economies. Previous research in this area was extended by the use of more robust traditional econometric models as well as the use of non-parametric specifications. Based on the results of these tests, it is shown that, in most organizations, the software development production process first exhibits increasing returns to scale, but that decreasing returns set in for very large projects. A method for determining the most productive scale size for any given organization is also shown. This research is an example of how the tools of economic analysis can be used to both test hypotheses about the effects of technological change and to provide software development managers with practical advice on how to increase productivity.

A second research effort primarily relates to Galbraith's second and fourth hypotheses, which both concern the factors of production. In particular, he suggests that capital will be substituted for labor, and that manpower requirements will become more specialized. These hypotheses present a number of challenging research questions. Can software engineering tools be used to replace manpower on software development projects? What is the extent of the learning curves for these tools? Have increased maintenance requirements changed the demand for tools or for staffing? What aspects of manpower specialization affect productivity?

Some recent research has begun the process of answering these questions for the maintenance phase of the software production process [Banker, et al., 87]. In this context maintenance means all of the activities following implementation, and therefore includes adaptive and perfective maintenance as well as corrective maintenance [Lientz/Swanson 80]. Data were collected on 65 software maintenance projects completed over an 18 month period at a large commercial bank. These data included the number of hours worked on the projects, the number of Function Points [Albrecht/Gaffney 83] and source lines of code produced, and the presence or absence of a number of environmental factors believed to affect productivity. A production process model was created, using the data envelopment analysis tool [Banker, et al. 84]. This model produced efficiency ratings for each of the projects. These ratings were then analyzed in light of 16 environmental factors believed to affect productivity. The results of the study showed that six of the environmental variables had a statistically significant impact on the productivity of the projects. These effects can be grouped into two categories that correspond to Galbraith's second (tool usage) and fourth (staff requirements) hypotheses, and a third category that can be called project management.

In the category of tool usage, under hardware tools it was found that better response time was significantly associated with higher productivity. This is consistent with the findings of a number of other researchers [Lambert 84, Thadhani 84], but is of broader relevance since it is not based upon sub-second response time. A methodological tool-related finding was that the use of a particular structured analysis and design methodology was negatively associated with productivity. This result is believed to be highly dependent upon the fact that productivity was being measured only at the level of the current project (as opposed to the long term productivity associated with an application) and to the fact that the methodology was newly installed at the site. What this result suggests, however, is that measurement of the impact of tools on project productivity will be very difficult, due to the downstream nature of some of the benefits and to the often significant learning costs involved.

The second category, staffing requirements, held the most statistically significant results of the research. A first, and intuitive result was that project teams composed of at least half "top performers" (as evidenced by their personnel ratings) were significantly more productive than teams not so composed. A second, and more novel, result was that team members' experience with the application was more important in explaining variations in their productivity than was their level of systems experience. Of course, this result may reflect the maintenance nature of the tasks, and only additional research may show whether this result holds in general. Nonetheless, this result suggest that increased specialization as suggested by Galbraith has already occurred along the application dimension. The managers at the research data-site acted upon this information by increasing the minimum amount of time that staff members were associated with an application before being rotated to other assignments. This is another example of the potential of the economic analysis tools to provide practical managerial recommendations.

Finally, a third group of significant factors was related to the management of the project schedule. Projects where the manager felt that there was greater than average deadline pressure were more productive than those where this was not the case. In addition, project loading (the ratio of work-months to calendar months, or roughly average staff on the project) was inversely related to productivity. That is, adding a lot of staff in order to complete a project lowered productivity, as suggested by Brooks's Law [Brooks 75].

In summary, the results of the scale research and the maintenance productivity research suggest both the relevance of the Galbraith technology hypotheses and the utility of the economic analysis tools employed. The next section discusses further research that is either ongoing or planned that will continue to build on the economic models and tools approach.

V. Ongoing/Future Research

There are a large number of possible research questions suggested by the economic production process model of software development. In this section several of these questions will be posed to further demonstrate the applicability of this model to a number of questions of theoretical and practical concern.

One question that arises as a direct result of the research described in section IV relates to the most productive scale size. In the organizations studied, the most productive scale size varied, and an interesting further research question would be to determine the factors that enable some organizations to manage larger projects successfully. Increasing organizations' ability to do this would have significant impacts on the profession, and could also help to mitigate any negative impacts of the hypothesized increase in the scale size that is predicated upon the use of technology.

One feature of the early research in software production processes is that simple, one input, one output production process models have been used. For example, the seminal work of Boehm uses labor, measured in man-months, as the input and a source lines of code metric as the output [Boehm 81]. This approach, while a reasonable surrogate for the process, may not fully account for the differences between the different phases of the system lifecycle.[3] A more complex view of the process would involve measuring the inputs and outputs at each of the stages, including requirements analysis, systems design, coding, and testing and implementation. A number of questions relating to the tradeoffs between phases, and to potential interaction effects could be very relevant. The possible intervention of tools to link these phases more tightly or to shift the division of effort more heavily into other phases would be of great interest to tool developers and users. And, the development of non-coding metrics for measuring these other phases would be of general interest.

Some current research [Banker/Kemerer 87b] is developing a two phase model involving both an analysis/design phase and a coding/testing phase. Of interest is whether a) the economy of scale results shown in previous research can be further substantiated with this more sophisticated model, and b) whether the two phases

[3]Note that Boehm's detailed version of his COCOMO cost estimation model allows for weighting of the "cost-drivers" by phase.

can be shown to be separable. Separability is a notion used by economists to characterize the interaction effects of two or more production processes. A manufacturing example might be two assembly lines, where one question is whether an increase in the volume of production on the first line has any impact on the productivity of the second. For software development it is of interest whether a large design effort has any effect on the effort involved in the coding phase, over and above that indicated by the size of the project. (i.e., A large project is likely to have both a large design and a large coding effort. The question is, whether, *ceteris paribus*, a large design effort has some impact on coding.) On the one hand, a large design effort may indicate that a thorough design was created, which should improve the productivity of the coding effort. On the other hand, if the staff members who are doing the design work are also doing the coding, some exhaustion effects may set in on large projects.

Research more directly related to the second Galbraith technology hypothesis would be to directly investigate the marginal rate of technical substitution (MRTS) of capital for labor in the software development context. Assuming that certain software engineering tools can actually be used to reduce the amount of professional staff time devoted to a project, then an economic production function could be developed that represented the relationship between the tool (capital) and the staff time (labor) that resulted in equal amounts of software being developed (i.e., an isoquant). With this function an obvious question of interest would be the optimal mix of capital and labor. Note that the answer is not necessarily simply the solution with the least amount of labor, since the cost of acquiring and using the tool (the factor price of capital) is significantly different from zero. Therefore, in order to determine the optimal capital/labor mix, it would be necessary to determine the rate at which the tool substituted for labor, the MRTS.

Given the possibility of decreased flexibility suggested by Galbraith's third hypothesis, there are a number of research questions to be addressed. First, how can the existence of this inflexibility be confirmed? Research into the diffusion of technological innovations may be of some help in this regard. Research should be directed at the three factors that are likely to cause this inflexibility, *de facto* standardization, software maintenance inventory, and learning curves. The existence and presumed impact of *de facto* standards is looming as an increasingly important research problem. The work of Sirbu and Stewart [Sirbu/Stewart 86] in the market for modems may be able to provide insights into a research model for software.

Software maintenance is a huge and growing problem, and some initial research directed at understanding how to increase productivity in this area was described in Section IV. Additional research, particularly at other sites, will shed further light on this phenomenon and should help practicing software managers reduce their costs in this area. As suggested by the research in software maintenance cited earlier, the learning curves involved with various software engineering innovations are likely to be non-trivial, and therefore may form serious obstacles to the adoption of software engineering tools. Research should be directed towards first measuring these costs, and then, based on the knowledge generated by the models of the process, suggest means of reducing them. This research would have significant direct payoffs in terms of the earlier and more widespread adoption of many of the new developments in software engineering.

In summary, numerous important research questions are both suggested by economic theory and are amenable to analysis using economic modeling tools. A key element to the success of these efforts will be the availability of high quality data from real world implementation sites.

VI. Concluding Remarks

This paper has demonstrated that the tools of microeconomics can be used to model software development as a production process. The six hypotheses of Galbraith's Technology Imperative Model were shown to provide significant insights into the likely effects of the increased use of technology in software development. The hypotheses from such a model can be empirically tested, and current examples of such research were presented. This research into scale economies and software maintenance productivity demonstrated the validity of many of the research hypotheses, and resulted in practical recommendations for software development management. These results are a direct outcome of the analytic tools, such as data envelopment analysis, provided by the microeconomic production process model approach.

Finally, a substantial list of additional research questions stemming from the microeconomic production process model were presented, with suggestions on how to continue the work begun in this area. The benefits from such research include not only a greater intellectual understanding of the software development production process, but also some immediately valuable recommendations to practicing software development managers. It is suggested that increased efforts be made to collect the empirical data necessary to apply the theories and tools of microeconomics to software engineering.

Bibliography

[Albrecht/Gaffney 83]
Albrecht, A. J. and J. Gaffney, Jr. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Transactions on Software Engineering, V. SE-9, N. 6, (November 1983), pp. 639-648.

[Banker, et al. 84]
Banker, R., A. Charnes and W. Cooper. "Some Models for Estimating Technical and Scale Inefficiencies in DEA" Management Science, V. 30, N. 9, (September 1984) pp. 1078-1092.

[Banker, et al. 87]
Banker, R., S. Datar, C. Kemerer "Factors Affecting Software Maintenance Productivity," Proceedings of the 8th International Conference on Information Systems, Pittsburgh, PA, Dec. 1987.

[Banker/Kemerer 87a]
Banker, R. and C. Kemerer "Scale Economies in New Software Development," Submitted to IEEE Transactions on Software Engineering.

[Banker/Kemerer 87b]
Banker, R. and C. Kemerer "Test of a Production Economics Model of New Software Development: Comparison of Parametric and Non-parametric Approaches," working paper, 1987.

[Bayer/Melone 87]
Bayer, J. and N. Melone "Predicting Acquisition and Adoption of Software Engineering Innovations," Carnegie Mellon University (GSIA) Working Paper #42-86-87, June 1987.

[Boehm 81]
Boehm, B. Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ (1981).

[Brancheau/Wetherbe 87]
Brancheau, J. and J. Wetherbe "Key Issues in Information Systems Management," MIS Quarterly V. 11, N. 1 (March 1987) pp. 23-45.

[Bratman/Court 75]
Bratman, H. and T. Court "The Software Factory," Computer (May 1975) pp. 28-29.

[Brooks 75]
Brooks, F. The Mythical Man-Month, Addison-Wesley, Reading MA (1975).

[Conte et al. 86]
Conte, S., H. Dunsmore, and V. Shen Software Engineering Metrics and Models, Benjamin-Cummings, Reading, MA (1986).

[Cusumano 87]
Cusumano, Michael "The Software Factory Reconsidered," MIT Sloan School Working Paper #1885-87, June 1987.

[DeMarco 78]
DeMarco, Tom Structured Analysis and System Specification, Yourdon, Inc. NY, NY (1978).

[Farrell/Saloner 86]
Farrell, J. and G. Saloner "Installed Base and Compatibility: Innovation, Product Preannouncements, and Predation," American Economic Review V. 76 N. 5 (Dec. 1986) pp.940-955.

[Galbraith 85]
Galbraith, John Kenneth The New Industrial State, 4th Edition, Houghton-Mifflin, Boston (1985).

[Grammas/Klein 85]
Grammas, G. and J. Klein "Software Productivity as a Strategic Variable," Interfaces V. 15 N. 3 (May/June 1985) pp. 116-126.

[Jensen]

Jensen, R.W. "Projected Productivity Impact of Near Term ADA Use in Software System Development," Mimeo, Hughes Aircraft, Fullerton, California 92634.

[Jones 86]

Jones, Capers "The Productivity Report Card", Software News V. 6, N. 9 (September 1986) p. 19.

[Kemerer 87]

Kemerer, Chris F. "An Empirical Validation of Software Cost Estimation Models," Communications of the ACM, V. 30 N. 5 (May 1987) pp. 416-429.

[Lambert 84]

Lambert, G. N. "A Comparative Study of System Response Time on Program Developer Productivity", IBM Systems Journal V. 23, N. 1, (1984) pp. 36-43.

[Lientz/Swanson 80]

Lientz, B. and E.B. Swanson Software Maintenance Management, Addison-Wesley, Reading, MA (1980).

[Manley 85]

Manley, John "Software Engineering Provisioning Process, "Proceedings of the 8th International Conference on Software Engineering (1985) IEEE Computer Society Press, pp. 273-284.

[Miyazaki/Mori 85]

Miyazaki, Y. and K. Mori "COCOMO Evaluation and Tailoring," Proceedings of the 8th International Conference on Software Engineering (1985) IEEE Computer Society Press, pp. 292-299.

[Parikh 85]

Parikh, Girish "Software Maintenance: Penny Wise, Program Foolish," Computerworld (Sept. 23, 1985) pp. ID11-ID16.

[Rubin 85]
Rubin, Howard A. "A Comparison of Cost Estimation Tools (A Panel Session)," Proceedings of the 8th International Conference on Software Engineering (1985) IEEE Computer Society Press, pp. 174-180.

[Schwartz 87]
Schwartz, David P. "Software Evolution Management: An Integrated Discipline for Managing Software," Proceedings of the 9th International Conference on Software Engineering, Monterey, CA (1987).

[Sirbu/Stewart 86]
Sirbu, M. and S. Stewart "Market Structure and the Emergence of Standards," Carnegie Mellon University Engineering and Public Policy Working Paper (1986).

[Spector/Gifford 86]
Spector, A. and D. Gifford "Case Study: A Computer Science Perspective of Bridge Design," Communications of the ACM V. 29 N. 4 (April 1986) pp. 267-284.

[Thadhani 84]
Thadhani, A. J. "Factors Affecting Productivity during Application Development," IBM Systems Journal V. 23 (1984) pp. 19-35.

[Walls 86]
Walls, J. G. "Information Systems Development: Product or Service?" Panel #7, Proceedings of the 7th International Conference on Information Systems San Diego, CA (Dec. 1986) pp. 347-348.