AN AGENDA FOR RESEARCH IN THE MANAGERIAL EVALUATION OF COMPUTER-AIDED SOFTWARE ENGINEERING (CASE) TOOL IMPACTS

Chris F. Kemerer

MIT Sloan School of Management
Cambridge, MA

ABSTRACT

While sales of computer-aided software engineering (CASE) tools are already significant and are continuing to rise, many software developers have adopted a 'wait and see' attitude, due in part to the complete lack of validated research demonstrating any of the tools proposed productivity benefits. This paper suggests that the lack of results in this area is due to a number of inherent difficulties in the CASE evaluation process, and proposes a series of research initiatives to address the shortfalls in managers' current ability to evaluate these tools.

I. INTRODUCTION

Sales of computer-aided software engineering (CASE) tools were over $100 million in 1987, and are expected to double in each of the next three years, according to a report by Business Technology Research [Case88]. The tremendous growth of this nascent industry has been fueled entirely by the overwhelming demand by practitioners for any remedy to the software productivity and quality crisis. However, how many more software developers have yet to adopt the tenets of the CASE faith, fearing that, like fourth generation languages or structured software development before it, CASE may be another unproven productivity nostrum? This evaluation problem is faced by practicing software development managers, but is currently being addressed by software engineering researchers and CASE tool vendors. However, efforts by both researchers and vendors have generally failed to validate the impacts of CASE tools. This problem stems from a number of inherent difficulties in the CASE evaluation research problem and in the limitations of the current approaches to evaluation.

This discussion paper proposes a list of research problems that 1) suggest some reasons why no significant results have been found by previous research and 2) set out an agenda of CASE research problems that require attention before substantial progress can be made on solving questions of evaluation. Solutions to these problems will, in the short run help practicing managers evaluate CASE technologies and in the long run will advance research in the general area of information technology impacts.

Broadly construed, CASE evaluation research problems reside in three areas: 1) impacts and their measures, 2) models, and 3) methodologies. (Clearly, there can be overlap in those three areas, in that the choice of a model may dictate a certain type of measure or suggest a particular methodology. However, for presentation purposes it will be useful to separate these categories.) The types of impacts that will be considered are those that are likely to be of interest to practicing IS managers, and are often (but not exclusively) those that are touted by CASE vendors in their marketing literature, including increases in systems development productivity and quality. By a model what is meant is a representation, possibly formal, of the software development process which allows the identification of the impacts of CASE tools. A measure is a particular metric for some construct in the model chosen. Thus, for example, a choice of source lines of code (SLOC) per work-month as a measure of the productivity impact of a CASE tool is based on some model, perhaps an economic production process model of capital for labor substitution.

These models and measures are realized or implemented by use of some research methodology, such as a formal experiment or a case study. In some circumstances the choice of methodology follows naturally from the research model and measures (e.g. a survey of systems developers' attitudes to investigate the impact of CASE tools on job satisfaction), whereas in other circumstances the researcher may have multiple options regarding how to best investigate the phenomenon at hand, such as collecting a large empirical dataset for statistical analysis versus running an experiment to research productivity impacts.

The format of this paper is as follows. Section II describes managerial impacts that are commonly associated with CASE tools and discusses why measuring these impacts can be difficult. Additionally, some less commonly cited impacts are raised and suggestions for research in these areas are made. Section III discusses the importance of models of software development to research in this area, and highlights the need for further work at this level. Section IV describes three popular research methodologies, experiments, field studies, and surveys and shows their respective limitations with regard to

0073-1129/89/0000/0219$01.00

CASE tool evaluation. Finally, Section V offers a brief summary and concluding remarks which prioritize many of the research problems presented earlier in the paper.

II. CASE IMPACTS AND THEIR MEASURES

Managerial impacts of CASE tools can be categorized along a number of dimensions. One broad categorization used in the paragraphs that follow is between those impacts that are most commonly ascribed to CASE tools, such as improved productivity and quality and those less widely discussed, such as the impacts on the quality of work life. These latter impacts can be divided into individual impacts, (such as whether or not CASE tools act to "deskill" certain occupations), and group or organizational impacts, such as power or status shifts. Finally, the measurement of the CASE tools themselves is discussed as a current impediment to research in all of these areas.

A. Productivity Metrics

The most commonly used productivity metric, source lines of code (SLOC) per labor unit (usually work-hours or work-months, depending upon the size of the project), suffers from two distinct sets of problems. The first is the general difficulty in capturing comparable SLOC metrics, a problem that has received extensive discussion elsewhere [Jone86] and will only be summarized here. The second problem, and more relevant to the CASE tools discussion, is getting agreement on whether increasing this ratio is desired, and how it can be used to assess new CASE tools that directly affect the numerator by changing the language level (for instance, in fourth generation-like very high level languages) or by automatically creating the SLOC, as in code generators.

The first set of problems with the SLOC per labor unit metric includes 1) varied counting rules for the numerator, 2) effect of changed or deleted lines, and 3) counting rules for the denominator. Jones has documented some eleven "standard" SLOC counting rules, mostly reflecting variations on rules concerning comments, line delimiters, and JCL [Jone86]. This problem is exacerbated by attempts to compare SLOC across programming languages. Rules regarding the proper procedure to count changed or deleted SLOC are becoming especially critical now that research attention is finally beginning to focus on software maintenance in addition to new software development. Finally, there is little agreement on how labor should be accounted for, with variations arising from what project phases are included, which staff personnel are included (e.g. all or just exempt, all exempt or just programmers), and which if any levels of management are counted. Worse than the fact that there is little standardization in these areas is that published research often neglects to inform the reader which decision about data collection were made.

CASE tools provide special measurement challenges in that many of the standard approaches, such as SLOC per labor unit are directly affected by the impact of the tool itself. For example, code generators produce lines of code at a rate hundreds or even thousands of times faster than a programmer writing by hand. Given that the current state of the art in these tools generally leaves some code to still be written by hand, what might be of the most interest is the combined productivity of writing the additional lines and understanding the generated code. Presumably the productivity of a programmer maintaining code he or she wrote is higher than that of maintaining machine generated code. Whether this effect is permanent or diminishes with time according to some learning curve is an open research question.

A similar question could be asked about code restructurers, CASE tools that increase the degree of structure of existing programs. The intent is that, by restructuring code, potential defects will be eliminated, thus reducing the amount of corrective maintenance to be performed in the future. Also, a claim could be made that the code will be easier to understand for a new programmer who is about to perform maintenance on a program. However, veteran applications programmers are often reluctant to use such tools, in that by having the restructurer change their code they lose some of their insight into how the program works, and therefore lose some productivity on future changes. Also, some restructurers generate more code than the existing program had, and therefore increase the total SLOC investment that requires maintenance, further complicating attempts to measure productivity in this area.

The issue of the use of restructured code highlights a more general problem with productivity metrics based on labor units. Typically these units are expressed simply as "hours charged", with no note taken of the individuals who charged the hours. This ignores a large body of literature highlighting the significant differences in the productivity of individual systems developers [Sack68, Curt81, Bank87]. These differences in ability are compounded by differences in general experience, experience with a particular language or tool, experience with a particular application domain, or, in the case of maintenance work, experience with a particular piece of software. Many productivity effects may be either a) masked by the effects of missing experience variables, or b) contingent upon a particular set of variables, such as use by relatively trained or untrained staff members. This latter case is particularly relevant in the discussion about CASE tools, as the full impact on productivity may be contingent upon sufficient experience or training so as to reach a higher point on the learning curve. Or, conversely, the impact of structured design tools that provide a lot of documentation of the application may prove to have a large impact on relative application

novices, but do little to aid programmers with previous experience with the application.

The last reason why SLOC per labor unit measures are deficient is that they are of little or no use in evaluating CASE tools that are not involved in the programming phase of the systems development lifecycle, such as computer-aided design tools. The impact of these tools must be measured in some other manner. One proposal is to measure Albrecht's Function Points per labor unit. While Function Points have a clear advantage over SLOC in this type of task, they are no panacea. They are very labor-intensive to collect, and require a non-trivial investment in training in their measurement in order to expect consistent results. Further, their development in a classic data processing environment limits their applicability in other domains, such as real-time systems, and requires some translation when used with database-oriented, as opposed to file-oriented systems [Symo88]. Finally, Function Points are still a measure of system functionality as opposed to business functionality. John Henderson has noted that one impact of CASE design tools may be to reduce the size of the system (by elimination of redundancy) while holding the business functionality, possibly unmeasured by Function Points, constant[1].

Few standard alternatives to the use of Function Points as a measure of non-coding productivity have been developed. For the requirements analysis stage some suggestions have been made along the lines of observing the entire process and noting the number of issues covered, alternatives examined, or even the number of conflicts raised. These are interesting suggestions, but as of yet no generalizable metrics have emerged from this work. In the detailed design area, a number of suggestions (typically stemming from structured analysis and design or database design techniques) have been made, including DeMarco's BANG or metrics relating to data structure charts [DeMa82, Gold81, Wrig88]. These also show promise, but have yet to become widely adopted.

Finally, another impact that is commonly associated with productivity, but need not be, is time to market. With the increasing emphasis on the use of information systems for strategic advantage, the competitor who is first able to produce a system (perhaps aided in his development by CASE tools) may reap substantial rewards. While productivity and time to market are likely to be highly correlated, they need not be identical. For example, a large systems development project is likely to have a number of parallel activities, not all of which are on 'the critical path'. Increasing the productivity of these slack activities will not reduce the time to market. Or, another problem may be that productivity is increased, but insufficiently to meet competitive threats, if, say for example, the system was completed in 18 months rather than two years, but the competition was done in one year. If research was focused on the time to market question, rather than the productivity question exclusively, new opportunities for CASE tools might emerge.

[1]Personal communication.

B. Quality Metrics

Quality means a wide variety of things depending upon one's perspective, even in the relatively limited domain of software. Therefore, in order to keep the problem tractable, the discussion here will be limited to those aspects of quality that are often proposed as areas of impact of CASE tools. Typically, the suggested impact is one of aiding in the reduction of defects, where a defect is some aspect of a system that causes a failure, and where a failure is any departure of the results of the system from what is required [Musa87]. In this definition then, defects could be systems designs that did not include requested functionality as well as program bugs that cause missing or incorrect results.

The impact of CASE design aids on the quality of a system design is difficult to measure, since there seem to be few measures of the "goodness" of a design. This is a critical shortcoming, given the conventional wisdom that defects introduced during the analysis and design phases that are not found until the system is operational are more expensive to correct than those discovered at the same time, but that were introduced later in the life cycle.

The attempts to document both a) the efficiency of defect removal methods (perhaps aided by a CASE debugging tool), and b) the cost-effectiveness of such approaches rest on the ability to measure the number and severity of defects in a meaningful way. Unfortunately, the metrics for counting and classifying defects are in the early stages. And, even if good schemes existed for such collection, the perennial question is whether identifying a large number of defects in the testing phase says anything about the quality of the final product. One assumption is that it is good to identify large numbers of bugs in that all programs have some set of bugs, and that the more that are found indicates that fewer remain. Alternatively, the argument can be made that the number of bugs in programs varies tremendously, and that finding a large number of bugs simply indicates that there are large numbers to be found. DeMarco notes that program bugs are much like their insect analogues in that if, for example, a cockroach is seen in a restaurant the assumption is not that THE cockroach has been found, but rather that the place is probably infested with them [DeMa82]. In summary, the problems of the inadequate understanding of what the number of defects discovered means and the poor quality of the data collection in this area are related, in that

better empirical data would help to clarify which model of defects is more appropriate.

C. Work Force Impact

One area of potential impact that has not gotten much attention from tool vendors or software engineering researchers, but has occupied the minds of some sociologists is the impact of such tools on the systems development work force. These impacts can be at the individual level, the group level, or the organizational level.

Two contradictory hypotheses are proposed in terms of the impact of CASE technology on the work-lives of the individual systems developers. The first of these is that tools such as CASE act to "deskill" an occupation, removing the craft aspects of the profession and making it dull and routinized. This argument is reflected in the early complaints about structured programming, which stated that by following the dicta the programmer had no control over his or her 'art'. The second, and opposite hypothesis is that CASE tools relieve the programmer or analyst of such mechanical tasks such as checking a database for consistency, or checking a set of data flow diagrams for completeness, leaving more time for creative work.

The problem with attempting to demonstrate the truth of either of these hypotheses is that good measures of systems development tasks seem to be lacking, and therefore any before and after testing founders on not being able to determine exactly what tasks were done at either stage, and therefore how they compare. A further complication stems from the fact that the kinds of systems that are being built and the tools used change so rapidly that such comparisons would be difficult to do even if reasonable measures existed. This is a generic problem to this type of measurement in that the effect of tools may be to create differences in kind (systems are completed that would not otherwise be attempted) rather than differences in the degree of efficiency with which certain classes of systems are created.

An entire other set of impacts can be proposed at the group or organizational level. Implementation of CASE tools into the systems development process can radically change the roles of team members. For example, a tool may lower the level of technical knowledge that is required to perform some programming task. For example, the writing of JCL or an assembly code interface may be automatically generated by a tool utilized by one of the applications programmers. Without the tool, they may have had to allocate the task to one of the systems programmers or more senior applications programmers. Thus, in this example the tool acts to reduce the power and authority of the more technical staff. Another example is the use of a design tool that that generates standard detailed design and program specifications. This may enable the centralizing of a group of programmers who merely code from the specifications, whereas without the tool on previous projects these programmers were members of decentralized project teams.

These types of group or organizational impacts have not received much research attention, and therefore there is a real shortage of research instruments such as metrics of power and relative centralization or decentralization. However, organizations that implement CASE tools in search of other impacts such as improved individual productivity ignore these organizational impacts at their own peril.

D. Tool Measurement

The above sections have dealt with measures of the impacts of tool usage. While this is a critical focus, another problem area concerns the measure of the use of the tools themselves. Too often studies simply note that a tool was or was not used. This dummy variable formulation is likely to miss out much of what is believed to be important and relevant about the use of CASE tools. These issues are a) the applicability of a tool to a particular project, b) the extent of use of a tool, and c) the level of skill used in applying the tool. Each of these issues poses some difficult measurement questions.

The question of tool applicability is important in that a study that finds no effect for the use of a tool without taking the applicability into account does not differentiate between tools that provide little or no benefit in general with tools whose value may be found only in specific sets of circumstances. These circumstances may be rather obvious things, like a COBOL restructurer not being valuable on an ADA (TM, US DoD) programming project, to more subtle ideas like whether the scale of a project warrants the use of a tool. For example, many tools are designed to aid systems managers and developers in dealing with complexity. If the system being developed is relatively not that complex (where complexity might be measured in a number of dimensions, e.g. data structure complexity or algorithmic complexity), then a particular tool may have no benefit. Indeed, the learning and other overhead costs may introduce negative net benefits [Bank87]. Finally, a tool may not be applicable because of the particular staff involved, whose prior skills and experience either do not permit them to take advantage of what the tool has to offer or, in the less likely case, make the marginal return on the tool zero or negative.

Measuring the depth of use of a tool is another issue that has not been widely addressed. Previous experience with older tools, such as text editors suggest two ideas. The first is that the vast majority of commands and features are not used by the average user (the so-called 80/20 rule may apply here). The second, which is perhaps a corollary to the first, is that a few individuals will make great use of a tool, pushing it to its limits and greatly increasing their productivity in the process.

Therefore, a simple statement such as 'the tool was used' is likely to be very meager representation of what is actually taking place.

Finally, the impact of a tool is likely to be highly variable from its first use to its second and subsequent uses due to the learning costs that are incurred. In fact, the learning costs for most CASE tools are probably significantly large enough to effect a decrease in productivity on the first projects on which they are used. While this is a well recognized phenomenon, the influence this should have in terms of how the results of experiments ("the experimental group used the tool, while the control group did not") and non-longitudinal field studies should be interpreted does not always seem to be appreciated. What needs to be done are longitudinal studies where the effects of learning can be accounted for.

III. MODELS FOR IDENTIFYING CASE IMPACTS

The role of models in the evaluation of CASE tools is to guide the research and to provide a framework with which to assess the impacts. Additionally, formal models, through analysis or simulation can be used predict the impacts of CASE tools.

A. Typical problems stemming from lack of good models

At the heart of the difficulties in researching the impact of CASE tools is the lack of a generally accepted underlying model of the software development process. Even the selection of a dimension or dimensions of impact is confounded by the lack of well-accepted definitions for such basic concepts as productivity and quality. Is a particular CASE tool going to make systems development more efficient by increasing productivity, more effective by increasing the quality, or both of these goals simultaneously?

Productivity and quality are closely interrelated, in that activities that benefit quality (e.g. defect removal) also improve productivity in the long term view of productivity, where the systems development lifecycle includes systems operation and maintenance. However, productivity is often thought of in the short term, or project sense, and therefore activities such as design walkthroughs and code inspections may appear to reduce project productivity where that is defined as completing the project with the minimum amount of resources. Therefore, whether increased quality comes with increased productivity or at its expense is very dependent upon the particular productivity concept employed.

Even when a definition of productivity is agreed upon, the lack of well-accepted models of the software development process hinders the ability to evaluate new methodologies and technologies. One familiar example is the set of techniques known as structured analysis, design, and programming. The notion is that careful development according to an explicit set of rules will result in "better" systems. Better has been taken to variously mean higher quality and/or easier to develop or maintain. However, a review by Vessey and Weber of about a decade's worth of research on the analogous issue of structured programming suggests that the results of research using a wide variety of methodologies have found very weak support for this contention [Vess84]. And given that there are some suggestions that the initial cost of development using these techniques is higher, it is no wonder that the implementation and acceptance of these techniques by practitioners has often been half-hearted.

Even when the constructs are well understood, the problem of inferring causal relationships from statistical data is very difficult. To cite one famous example, the Atlantic Systems Guild supported a study called "Coding Wars", where programmers from a number of field sites coded a given program and recorded the time it took them, along with a number of other environmental factors [DeMa85]. One of the results of the study was that programmers who had a) larger offices and b) secretaries to pick up their ringing phones were more productive. This begets the question of whether bigger offices make better programmers, or whether better programmers get promoted and get bigger offices.

A more common real life example arises due to how project teams are assigned. Presumably, more senior staff with greater experience are more productive and produce higher quality systems than newer staff members. However, these senior staff members are also likely to be assigned to manage larger and more complex projects, both due to their greater ability to handle the responsibility and as part of the natural promotion process. Therefore, studies of the effects of experience and ability may be masked by the assignment of these better individuals to more difficult projects.

B. Goal driven vs process driven models

What these examples point up is that clear underlying models of the software development process have not been agreed to. One set of candidates are the "goal/product driven" models, such as a microeconomic production process model. In this model, software development is largely a black box where the inputs of labor and capital are transformed into tangible goods and services. The extreme case of this model is the "software factory" approach, where software is viewed as a product rolling off an assembly line [Cusu87]. This latter approach has been criticized as underrepresenting the service rather than product aspects of custom systems development [Coop87]. Also, the large degree to which the labor inputs are at the craft level rather than easily substitutable unskilled assembly line workers may cause sufficient anomalies to upset the assumptions of the factory model. Other models of software

development are process-driven, and concentrate on the human aspects, particularly the relative power wielded by systems developers and systems users. These models can be criticized as not taking sufficient account of the influences of the technical aspects of systems development, with the argument being that systems development is more than simply a negotiation problem like others in the organization. What are really needed are models that integrate these two perspectives, product and process. Until there is greater agreement on these types of modeling questions, progress in the evaluation of individual aspects of the software development process will be held back.

IV. METHODOLOGIES USED TO EVALUATE CASE IMPACTS

The preceding two major sections have suggested that analyses of the impacts of CASE tools are negatively impacted by the lack of agreement on models of the likely effects of the tools and by the absence of robust measures of both the impacts and the tools themselves. This section will discuss the limitations imposed by current methodological approaches to research on CASE tools. The approaches generally consist of a) formal experiments, b) single site field studies, or c) multi-site surveys.

A. Formal Experiments

In the experimental approach, some attempt is made to control for the effects of variables other than the variable of interest, here assumed to be use of CASE tools. The difficulties with this approach stem partially from general factors that plague all social science research using this approach, and partially from factors that are more specific to the systems development and CASE tool problems.

The general problem with this type of work is that the overall assumption of *ceteris paribus*, all other things being equal, is extremely difficult to enforce. The problems of knowing which other variables, such as experience, ability, and interests are important for the experimental task, and then knowing how to measure and control for them (perhaps by matching) is difficult. Other problems, such as the Hawthorne effect, also affect experiments with human subjects.

Added to these general problems are the particular problems imposed by the systems development problem domain and the measurable impact of CASE tools. One issue is that due to the scarcity of systems development talent relative to the demand, developing systems is a very expensive proposition, which makes systems development staff too valuable a resource to use in large experiments. Second, and related to this imbalance between systems demanded and the current rate of supply, is that systems development management is often so overwhelmed by the short-run concerns of getting the current workload accomplished that their interest in work with long term payoffs, such as research, is often limited. Finally, the expense of such experiments militates against having the large sample sizes that could help to alleviate the problems associated with human variability.

One research reaction to this is to attempt to measure the actual results of a project using the CASE tool, and then ask the developers to estimate what the project would have required without the tool [Lemp88]. The difficulties with this approach are apparent if consideration is given to the fact that there is ample research to suggest that software developers are very poor at effort estimation [Keme89]. Therefore, while this approach has been a reasonable one for previous exploratory work in CASE evaluation, future research needs to develop approaches that lend themselves more readily to independent validation and confirmation.

The other set of research reactions to these problems has been either a) to use students as substitute subjects for professional systems developers, b) to select very small tasks for experimental subjects to perform, or c) both. The problem with student subjects is that their skill and experience levels are likely to be very different from that of the professional systems developers for whom the tools are intended [Cont86]. This suggests that the results from these experiments may not apply in the real world environment (a problem known as lack of external validity). This is a systems development problem rather than a problem with experiments generally since much research, in the psychology of decision-making, for example, is not generally impacted by factors such as particular real world job experience.

Even if real world subjects are used, because of their market value, experiments are generally constrained to very small tasks, given that the parallel design of typical experiments requires redundant effort. The specific problem here is that many, perhaps most CASE tools are designed to solve problems associated with programming-in-the-large. Their effects on small problems are likely to be limited, or perhaps even negative, given the fixed costs discussed in a preceding section. A final problem is that the impact of many CASE tools (for instance, structured design methodologies) may only have an impact on productivity over time, and therefore classic experiments are unlikely to capture these longitudinal effects.

What remains to be done in this area is for experiments to be directed only at those micro problems that, because of the perceived lack of longitudinal effects, are deemed appropriate to research via the experimental method. In addition, what would be helpful would be greater participation on the part of industry and government information systems managers to allow their staff to participate in small but well-run formal experiments. Some recent work at the Federal Software Management Support Center is an example of a step in this direction [FSMS87].

B. Single-site Field Studies

In order to overcome the economic obstacles surrounding formal experiments without resorting to student subjects, one practical alternative is to study actual projects at a real world site. While in many respects an improvement over the limitations of formal experiments, field studies have their own set of obstacles to effective CASE tool research. These are 1) finding acceptable sites ("data quality"), 2) getting sufficiently large sample sizes ("data quantity"), and 3) being able to draw appropriate inferences in the absence of an experimental structure.

B.1 Data Quality

In order for a field site to be acceptable, a number of criteria must be met. Primary among these is the ability to collect sufficient high quality data to make analysis possible. Unfortunately, the average commercial firm does not, on a routine basis, collect the information necessary to perform these kinds of analyses. Given some of the measures discussed in previous sections, it is clear why this is the case. Work-hour data, if captured at all by project, may reflect sloppy or even incorrect bookkeeping. Staff members or managers may not report hours if they will lead to a project going overbudget. Or, they may report them to another project or to an overhead account, further compounding the error. Even if accurate records are kept by project, detailed data by phase, by person, or both are kept only in a few firms. Unfortunately, the lack of such data prohibits the analysis of many of the most interesting proposed effects of CASE tools, such as the shifting of effort from mundane to more creative tasks, and the increased ability of junior personnel to handle tasks previously reserved for individuals with more expertise.

Even if labor hour data are correctly captured, the likelihood that all other data, such as data on output, defects, user satisfaction, complexity, and environmental factors such as system response time, are also captured decreases almost in direct proportion to the degree of detail required by the research question. In other words there are broad effects that are well known, such as that more experienced project teams are more productive, but attempts to get beyond these broad effects into the more interesting questions of how and why these effects occur require the type of detailed data that is rarely available.

B.2 Data Quantity

The problem of sample size also frustrates attempts to do good field research. While much of the data collection may already be in place within a progressive IS organization, undoubtedly a well done study will require other data that will require new collection efforts. Practitioners are understandably concerned about not placing any additional significant demands on an already overburdened IS staff. The need to minimize the impact of these additional data collection efforts may require the sample to be limited to a small group of hopefully representative projects. In order to reach a sufficient sample size, one of two options are likely. Either data from past completed projects can be used, in which case care must be taken to ensure the accuracy of the non-contemporaneous data, or the researcher must wait for data from future projects. Collection of historical data is often made particularly problematic by the high turnover in most IS departments. It is not at all unusual to begin to collect data on a completed project only to discover that the project manager or some other key individual no longer works there. This sometimes results in projects being excluded from the analysis, since the contemporaneous records require interpretation or supplementation from the project manager, whose absence renders this impossible.

There are two problems with waiting for future data. The first is that care must be taken to ensure the accuracy of data being collected by individuals who may perceive alternative uses (particularly managerial control uses) for the data. For example, self-reported data on SLOC or Function Points delivered may be "fudged" to give the impression of high personal productivity. The second problem is that it may take the typically-sized firm a long time to generate sufficient new projects from which to collect data. This problem is exacerbated in the evaluation of CASE tools since many of them are designed to be of greatest (perhaps any) value only on large systems. A firm is likely to do these large systems projects only infrequently, and of course, being large, they take a long time to complete.

The ultimate danger with making the research take a long time is three-fold. One, the field site is less likely to sponsor a research project at all whose results are more than a year away. Two, the pressure to publish discourages researchers from working on multi-year projects. Three, there is also a danger that if the results take very long, then changes in the business or the technology at the site will be sufficient so as to invalidate any results obtained. All of these problems also decrease the likelihood of any longitudinal research being done, a prospect that dims the possibility of observing some proposed CASE tool effects at all.

B.3 Validity

Problems of validity are serious obstacles to performing meaningful field research in CASE tool evaluation. CASE tools, like other new methodologies, are likely to be "rolled out" using one or a few specially selected pilot projects. These pilot projects may or may not be representative of the population of systems projects as a whole. If volunteers are solicited, then clearly there will be some selection bias on the part of rapid technology adapters or simply staff members who are dissatisfied with their current work. Even if pilot projects are

selected by management, the Hawthorne effect may still predominate. Therefore, it may be difficult to get a representative sample, in addition to the previously described learning curve problem.

Finally, the ultimate problem with any field study may be its external validity. Even if a CASE tool effect can be shown, and a causal relationship is believed to be supported by the statistical data, there is little theoretical basis for assuming those results extend to other firms, even other firms that appear similar. A CASE tool implementation at one firm may have been well-received by a staff that is eager for its use, well-educated in its theoretical background, and gently introduced to its mechanics through excellent training and ongoing support. The same tool at another site may reach unreceptive ears, and be seen as being forced on them by management. These implementation differences are only one example of how firms may differ, and therefore how a tool can fail to have a similar impact elsewhere. Differences in applications mix, technical environment, personnel, management, users, backlog, organizational structure, and history can all have an impact on the final result. Therefore, field studies, while a critical part of any research portfolio, are limited in the total results they can support.

For all their problems and expense, field studies still seem to be a promising avenue for CASE tool evaluation research. What needs to be done are larger and better controlled studies and greater publication of data and results, so that other researchers and practitioners can determine the degree to which they feel the results will be valid in other environments.

C. Multi-site Surveys

Another approach that is less commonly applied to the study of CASE tool evaluation, but is still deserving of some discussion are multi-site surveys Multi-site surveys are sometimes employed to counteract the external validity questions raised by single site field studies. They can also be used to generate larger sample sizes, when it is believed that the different sites represent a single statistical population, and therefore can be pooled (For an example of such work, see [Norm87].).

Many of the problems of multi-site surveys are the same as the data problems encountered by single-site field studies, which are the lack of accurate and detailed data collection by the sites. This problem may even be compounded in a survey (typically done by mail) where there is little opportunity for the researcher to perform quality control, (if less than good quality data are recorded by the survey participant) with the exception of gross sensitivity analysis to comb out obvious outliers. Well-designed surveys can attempt to mitigate this situation, through the use of multiple informants, or through the use of multiple methods, but in practice this may still leave some unsolved problems. One possible approach in software engineering research is to use both an objective and a subjective measure of some construct such as complexity. The systems analyst or project manager will be asked to rate complexity on a scale, and, in addition, some measure such as McCabe's cyclomatic complexity will be recorded [McCa76]. The difficulty arises when these two metrics disagree. Does it mean that the objective metric (e.g., McCabe's) does not capture all of the notions of complexity, or is this yet another piece of evidence that humans are poor information processors? A conservative approach to research might be to discard those data points where there is a low level of correlation between the metrics. However, due to the difficulties discussed above in getting significant sample sizes, the temptation not to do this is great.

Another, perhaps more common use of survey data is in assessing attitudes towards software engineering innovations such as CASE tools. Many of the problems in the use of surveys to capture objective data are removed, but attitudes are generally of less interest than the other types of impacts that researchers attempt to measure. And even attitudinal measurement is far from easy. One problem with attitudes towards CASE tools is that the responses to initial surveys after a first implementation of CASE tools may not be reflective of attitudes after users are further along the learning curve. Or, reactions may simply reflect some standard inertia that is to be expected to any change in the established procedures. Ideally, surveys would be conducted before, after and much after the initial implementation of the CASE tool in order to more accurately assess the impact. However, this longitudinal approach to surveying suffers from the same obstacles to longitudinal research that have been previously discussed.

Surveys can be a useful research tool, but their use needs to be limited to areas where the costs of field studies cannot generate the necessary sample sizes, or where attitudes about the use of CASE tools are of interest.

V. SUMMARY: AN AGENDA FOR RESEARCH

This discussion paper has presented a long list of why CASE evaluation research is difficult. These problems center around the lack of well-understood models, the lack of widely accepted metrics, and the difficulty of applying standard research methodologies to CASE evaluation problems. These problems have left open the question as to whether the lack of substantial evidence for the impact of CASE tools has been due to a true lack of impact or whether the current research approaches are inadequate for the task.

However, in all problems the optimist sees opportunities, and the evaluation of CASE tools should be similarly viewed. Researchers wishing to make

contributions that will have an impact on practice as well as contributing to a theory of software development production have ample opportunities, as revealed by the lists below.

In the area of metrics, a number of topics are worth pursuing. In terms of existing productivity metrics more work needs to be done in terms of refining SLOC metrics to take account of the effects of changed, deleted, and generated code as opposed to the current assumptions of only handwritten new code. In addition, work should proceed on incorporating personnel aspects such as application or tool experience, and task aspects, such as the degree of structure of existing code for maintenance projects. In terms of new metrics, research on metrics for the analysis and design phases would fill a needed void, and additional work on quality metrics, particularly work towards developing standards in the collection of defect statistics would also be a major contribution. Metrics for tool usage, particularly functionality and extent of use are important levers for additional research. Measurements of tasks and the skill levels necessary or applied would do much to progress research on work force impacts.

The lack of well-understood models of software development has been shown to retard progress in the evaluation of CASE technology. Work should proceed along two paths. The first is theoretical work, perhaps relying on models from other disciplines. Production models from microeconomics and management science are one source for this work, as are process models from the behavioral sciences.

The second, and parallel set of activities should focus on empirical data gathering with which to validate and refine the theoretical models proposed. This empirical data gathering should rely on multiple methods for data collection, especially field studies but also including experiments and surveys where appropriate. Emphasis should be placed on committing resources for longitudinal studies, since this is where many of the CASE tool effects are likely to be present.

This is a large agenda for research, and will require the dedicated efforts of both academic researchers and practitioners. However, there is some room for optimism. Practitioners have acquired a heightened sense of the importance of collecting data on their systems development activities, thus preparing the way for future researchers [Kitc86, Grad87]. And researchers have profited with greater understanding owing to the initial attempts to establish results in this field. By addressing the problems stated in this paper, the research in the next decade should bring a series of results that will provide better insights for practitioners, vendors and the entire CASE tool community.[2]

[2]Helpful comments from participants at the MIT MIS Design workshop, W. Chismar, and three anonymous referees are gratefully acknowledged.

REFERENCES

[Albr83]
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 SE-9 (6):639-648, November, 1983.

[Bank87]
Banker, R., S. Datar and C. Kemerer
Factors Affecting Software Maintenance Productivity: An Exploratory Study.
Proceedings of the 8th International Conference on Information Systems, Pittsburgh, PA, December 1987.

[Boeh81]
Boehm, Barry W.
Software Engineering Economics.
Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Case88]
CASE Growth Will Skyrocket,
Software Magazine, V. 8, N. 3, March 16, 1988, p.16.

[Cont86]
Conte, S., H. Dunsmore and V. Shen.
Software Engineering Metrics and Models.
Benjamin/Cummings, Reading, MA, 1986.

[Coop87]
Cooper, Randy
Management Information Systems Development: Initial Thoughts Comparing Service Delivery and Goods Production Viewpoints
University of Michigan Working Paper, 10/21/87.

[Curt81]
Curtis, Bill.
Substantiating Programmer Variability.
Proceedings of the IEEE 69 (7):846, July, 1981.

[Cusu87]
Cusumano, M.
The Software Factory Reconsidered
MIT Sloan School Working Paper #1885-87, June 1987.

[DeMa82]
DeMarco, Tom.
Controlling Software Projects.
Yourdon Press, New York, NY, 1982.

[DeMa85]
DeMarco, T. and T. Lister.
Programmer Performance and the Effects of the Workplace.
Proceedings of the 8th International Conference on Software Engineering, Pages 268-272. IEEE Computer Society, Washington, D.C., August 28-30, 1985.

[FSMS87]
Parallel Test and Evaluation of a Cobol Restructuring Tool
Federal Software Management Support Center, United States General Services Administration, Office of Software Development and Information Technology, Falls Church, VA, September 1987.

[Gold81]
Golden, J. R., J. R. Mueller and B. Anselm.
Software Cost Estimating: Craft or Witchcraft.
Database 12 (3):12-14, Spring, 1981.

[Grad87]
Grady, R. B. and D. L. Caswell
Software Metrics: Establishing A Company-Wide Program
Prentice-Hall, Englewood Cliffs, NJ (1987).

[Jone86]
Jones, Capers.
Programming Productivity.
McGraw-Hill, New York, 1986.

[Keme87]
Kemerer, Chris F.
An Empirical Validation of Software Cost Estimation Models.
Communications of the ACM 30 (5):416-429, May, 1987.

[Keme89]
Kemerer, Chris F.
Software Cost Estimation Models
Forthcoming in *Software Engineers Reference Book*, Butterworth Scientific, Ltd., Surrey, England, UK (1989).

[Kitc86]
Kitchenham, B. A. and J. A. McDermid
Software Metrics and Integrated Project Support Environments
Software Engineering Journal, v. 1 n. 1, January 1986, pp. 58-64.

[Leav87]
Leavitt, Don.
Scratching the Surface of CASE Potential.
Software News 7 (2):50-53, February, 1987.

[Lemp88]
Lempp, P. and R. Lauber
What Productivity Increases to Expect from a CASE Environment: Results of a User Survey
Proceedings of the ACM 27th Annual Technical Symposium, Gaithersburg, MD, June 9, 1988.

[McCa76]
McCabe, T.
A Complexity Measure.
IEEE Transactions on Software Engineering SE-2:308-320, December, 1976.

[Musa87]
Musa, J., A. Iannino, and K. Okumoto
Software Reliability
McGraw-Hill, NY, NY (1987)

[Norm87]
Norman, Ronald
Integrated Development Environments in Support of Information Systems Design Methodology and Systems Analysts' Productivity, PhD. Dissertation, University of Arizona, (1987).

[Sack68]
Sackman, H., W.J. Erikson, E.E. Grant.
Exploratory Experimental Studies Comparing Online and Offline Programming Performance.
Communications of the ACM 11 (1):3-11, January, 1968.

[Symo88]
Symons, C. R.
Function Point Analysis: Difficulties and Improvements
IEEE Transactions on Software Engineering, v. 14, n.1, January 1988, pp. 2-11.

[Vess84]
Vessey, I. and R. Weber
Research on Structured Programming: An Empiricist's Evaluation
IEEE Transactions on Software Engineering SE-10 (4):397-407, July 1984.

[Wrig88]
Wrigley, C. and A. Dexter
A Model for Estimating Information Systems Requirements Size: Preliminary Findings
Forthcoming in *Proc. of the 9th International Conference on Information Systems*, Minneapolis, Minn. December 1988.