THE BOUNDING EFFECT OF IS DESIGN TOOLS: A CRITICAL EXAMINATION OF CASE TECHNOLOGY

by

Gad Ariav
Leonard N. Stern School of Business
Information Systems Department
New York University
100 Trinity Place
New York, NY 10006

and

Wanda Orlikowski
Sloan School of Management
Information Technologies Group
Massachusetts Institute of Technology
50 Memorial Drive
Cambridge, MA 02139

March 1989

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

CRIS #204
STERN #89-36

The authors are listed alphabetically.

The Bounding Effect of IS Design Tools: A Critical Examination of CASE Technology

Abstract

Methodologies for information systems development bound the vocabulary of design (what are the "things" that matter?), as well as control the design discourse (how should we go about discussing them?). Computer Aided System Engineering tools -- collectively refered to as "CASE technology" -- further bound the analysis and design process both semantically (e.g., the range of available methodologies) and syntactically (e.g., implemetation details). In this paper we explore the effects of bounding in CASE technology. We first delineate the concept of bounding in general terms, and then develop a more operational notion of it through the qualitative examination of an actual use of a CASE tool. This examination results in a preliminary list of concrete dimensions of the bounding phenomenon, which is in turn used to guide a critical survey of related features in current CASE technology. Implications for practice, education and research are discussed.

1. Introduction

The topic of computer aided software engineering (CASE) features prominently in the contemporary agenda of the information systems (IS) community. Nevertheless, the impact of CASE technology is still limited -- it is estimated that only 7 or 8 percent of the programmers in the U.S. have been exposed to these tools [2]. These numbers will no doubt change -- it is estimated that the entire installed base of CASE systems has more than doubled during 1988 [4]. Common wisdom also suggests that the cummulative attention currently given to the topic and the imminent entry of IBM to the CASE arena with its Repository system will significantly amplify the actual effect of this technology. A thorough assessment of the potential impact of CASE tools is therefore urgent.

This paper examines the effect of CASE technologies on the process of information system design. Insight into the effect of CASE proves to be doubly elusive, due to the relationship between this technology and IS design in general. Specifically, while we still try to gain a better grasp of the process by which information systems are designed, CASE tools are already attempting to automate it, or at least some parts of it. As any textbook on system analysis and design rightfully argues, automation of a poorly understood task is at best a risky prospect. A deeper understanding of the impact of CASE technology therefore ultimately hinges on our understanding of the IS design process at large. In particular, final assessment of the significance of the results reported in this paper can only be made in the context of the broader discussion of the effect of IS design methodologies, of which CASE tools are a proper subset.

This study approaches a core issue in the design process -- the design methodology -- from a different perspective. Attention typically focuses on the methodology's substantive content

-- the guidelines and imperatives it prescribes in order to facilitate the process of design. Fundamentally, however, a design methodology is an agenda-setting mechanism, and following any methodology implies subscribing -- knowingly or unknowingly -- to a corresponding **bounding** of the design discourse. The concepts of "methodology" and "bounding" in the context of IS design are the flip sides of the same design experience. They differ mainly in their emphasis -- the former emphasizes the "positive" aspects of inclusion, i.e., what to focus on in a system study, while the latter emphasizes the complementary "passive" aspects of exclusion, i.e., what has been left out.

Examining a methodology directly or examining its bounding effect are both a study of the process of design, yet from diametrically opposing vantage points. The "exclusion problem" is the dual problem of the "inclusion problem," and therefore their respective solutions are conceptually equivalent. If done properly both approaches should yield consistent results, although there are typically differences with respect to insight and convenience. In that respect, studying the bounding effects highlights some of the obvious -- and typically unnoticed -- effects of IS design methodologies. Under this "duality view" the study of bounding effects ties in with the long tradition of interest in understanding processes of design and IS design in particular. Examining bounding effects is in essence a shift from the "foreground" to the "background" in studying the design process. While this equally applies to the study of any design methodology, adopting this perspective in the study of CASE technology is especially appropriate: CASE tools quintessentially cast methodological choices in a more concrete structure, and make their limiting nature more acute.

Since the study of bounding is relatively new, the structure of this paper reflects a careful attempt to develop a valid operational framework for examining the bounding effects of CASE technology. Specifically, Section 2 briefly delineates the notions of bounding and CASE technology. Section 3 identifies actual and concrete manifestations of bounding effects as they emerged from empirical evidence gathered at a business site where a CASE tool is being used extensively. In light of these empirical observations Section 4 critically reviews the current state of CASE technology through a comprehensive survey of bounding features in contemporary CASE products. Section 5 places the previous discussions in a broader perspective, and considers the major findings in terms of their impact on information systems practice, education and research.

2. The Nature of Bounding in CASE Tools

In this section we first review the nature and current perceptions of CASE technology, and then outline the concept of bounding in the context of CASE.

2.1. CASE Tools

The broad definition of CASE encompasses the use of information technology to support the various tasks of software engineering (SE). This definition is meant to outline the class of services addressed by these products, deliberately avoiding more refined statements of CASE which are typically aimed at excluding competitors' products from consideration[1].

As the variety of tasks collectively labeled "software engineering" is large, so is the variety of CASE tools. CASE tools obviously differ from each other with respect to their appearance, interaction style, range of services and specifics of implementation. In terms of their basic functionality, CASE tools can be classified along the following three dimensions:

1. SE Tasks Addressed: Which SE activities does the tool support? The range of possible activities starts with business analysis, continues with activities like requirements documentation, system specification, actual software construction, through system testing, conversion and installation.
2. The SE Process: Which generic SE processes does the tool suports? The four commonly referred to are system development, system enhancement, system migration, and system maintenance.
3. Extent of Integration: How well are the different parts of the CASE tool integrated? The range of possible approaches to product integration starts with support for a single task, through a family of loosely coupled, largely compatible set of tools, and ending with a fully integrated tool.

The common CASE tools address only few tasks, cater to the particular nature of software development processes, and are marked by only limited integration. The trend, though, is toward products that provide broader support, deeper integration, in a wider variety of SE processes.

CASE tools as we know them today were first introduced in the early 1980's. They are characterized by (1) a graphic interface for diagramming system specifications (data structures as well as processing logic), (2) a "repository" -- a.k.a. data dictionary, system encyclopedia, or design database -- where various system specifications are stored and maintained, and (3) a set

---

[1]As has been the case with other emerging information systems technologies, the area of CASE is currently still a "vendor province."

of algorithms that check stored system specifications for completeness and consistency [4]. The trend has been to augment CASE products with SE-related services like project management and code restructuring.

2.2. The Nature of Bounding

Bounding effects are inherent in any methodology, and hence inherent in every design methodology which fundamentally represents a vocabulary of design concepts and a pattern of design discourse. This inherent form of bounding we refer to as constitutional bounding to reflect the notion that the essential activity of design is constituted by a set of underlying assumptions, concepts, norms, interests, and values - in other words, a language [5]. This level of bounding is at the root of every formalized, disciplined approach to IS design. A comprehensive investigation of such bounding is beyond the scope of this paper. Instead we are interested here in the forms of bounding that are specifically implicated in design processes mediated by CASE technology.

In particular we wish to draw attention to two further levels of bounding. Methodological bounding recognizes that each specific CASE tool supports a different set of system design methodologies for the tasks that it addresses. Each tool thus limits the range and variety of design approaches that can be incorporated and drawn on in the design process. This second level of bounding is typically reinforced by organizational policy that mandates restriction to one or a subset of design methodologies in order to provide a uniform and consistent design platform within the organization. Implementation bounding reflects the specific constraints imposed on design activity by the physical implementation of particular CASE tools. These physical constraints affect the degrees of freedom offered to the designer/tool user with respect to the sequence of design attention, representation and manipulation of objects, interface characteristics, or possible methodological "short-cuts."

These three bounding effects form a hierarchy, and their effect is cumulative and nested. While all three levels of bounding - the constitutional, methodological, and implementation - have implications for both the semantic (content) and the syntactic (form) aspects of design, they vary in the extent of their influence. The constitutional and methodological bounding effects are most clearly seen through the semantic aspects of design activities. And these are accentuated through the mediation of CASE tools, as every such tool - implicitly or explicitly - subscribes to a

subset of design methodologies that constitute the conceptual or substantive context through which designers/tool users enact their designs. Implementation bounding effects have explicit and overt implications for the syntactic aspects of design, as it is at the level of implementation that CASE tools formalize and proceduralize specific execution paths that support the design process. The details of a specific tool implementation impose a context of use on the designer/tool user by determining the spatial and temporal conditions within which design tasks are executed. The form of the design activity is thus shaped by the particular technical manner and operational environment in which a CASE tool is instantiated.

The notion of bounding is still evolving, and a rigorous definition of it has yet to emerge. In lieu of such definition we develop in the next section an ostensive definition which highlights semantic and syntactic bounding effects. This definition draws on the actual experiences of a number of project teams using a specific CASE tool to develop information systems.

3. Evidence of Bounding Effects in The Use of CASE Technology

When assessing a CASE tool for its bounding potential, what aspects should be examined more closely? In this section we respond to this question by drawing on the findings of a study that investigated the role of a single CASE technology on a number of custom systems development projects [7]. Four large (over a hundred developers, 2-3 years duration, average of $10 million) and one small project (fifteen developers, less than a year's duration, a few hundred thousand dollars) were studied. The empirical data were collected via multiple methods employing extensive interviews, observation and documentation review.

The particular CASE product examined consisted of a family of capabilities operating as loosely coupled tools which were integrated through a central data repository (the project data dictionary) and a number of bridges that served as gateways between various tools, among them entity-relationship models, data flow and data structure graphic editors, data definition editors, text editors, screen and report design aids, support for reuse of standardized modules and shells, program code generators, macro library support, job set-up assistance, testing tools, and version control aids. The bounding effects of this CASE technology (henceforth known as ToolKit) were evident at both the semantic and the syntactic levels of systems development. Each will be discussed in turn.

3.1. Semantic Bounding Effects of ToolKit

The semantic bounding of CASE technologies refers to the effect of the repertoire of design concepts that given technologies support. ToolKit embodies the tenets of structured design [YourdonConst78] and supports the elaboration of program logic via Warnier-Orr data structures. Information requirements are determined by articulating the key inputs and outputs of a system, which are manifested via screens and reports. Databases and programs are defined around these inputs and outputs, with database designs employing Entity-Relationship modeling (no automated normalization is supported) and program structures being derived through Warnier-Orr data structures by augmenting existing generic program shells. Because this monolithic approach to systems design was the only approach supported through ToolKit, it was *de facto* the only acceptable design procedure employed on the projects examined.

The use of ToolKit on systems development projects in the organization was mandated so that analysts had little discretion over their use of the CASE technology. Because ToolKit had automated the adherence to a particular design methodology, developers were unable to contemplate alternative ways of approaching problems. The view of the "appropriate" design procedure which was embedded in ToolKit was clearly recognized as a source of bounding effects. Some of the more experienced analysts recognized that their attention had been restricted by ToolKit. A few comments from the field capture this:

> "With tools we force one path, and force everyone down that one path. I am not sure it's the right path, but at least it's a standardized path."

> "In the [design] stage there is too heavy an emphasis in the tools on the information system externals such as screens and reports. I think our focus should rather be on designing functions. But we tend to focus exclusively on the visual, tangible things, that is on the things that we can measure and count. And then these things become our measure or definition of the system we're developing. So for example we say our system is 200 screens big, not how many functions there are, or how complex they are. And we sign-off on screens and reports, not on functions. This affects the way you design your system and how you interact with users."

The ToolKit had an embedded methodology that directed the attention of analysts to certain aspects of the users' problem. In particular it defined techniques for eliciting requirements from users, prescribing the format of interview, nature of questions and method of data representation. Analysts translated these prescriptions into a "checklist" of questions that they used to interview users, hence prompting description of certain work experiences. In effect these questions were posed and structured so as to evoke those responses that could be represented in the data dictionary of ToolKit. Thus the restricted vocabulary of the design approach supported

in the CASE technology sensitized developers to ask specific, directed questions, focusing their attention on only certain aspects of the users' reality. The particular diagrams and design objects offered by the CASE tool are therefore another contributor to semantic bounding effects.

For example, ToolKit facilitated capturing design information on data, programs, screens and reports, via predefined forms that had been built into it. When analysts interacted with these forms, ToolKit prompted them for the specific design information that needed to be specified. A senior analyst suggested that this greatly influenced the requirements determination work in the early conceptual design stage:

> "I believe in separating functions from the images that the functions reflect to the outside world, the screens and reports. But the tools make us talk to users about their inputs and outputs, their screens and reports. The argument is that users don't understand functions; that they're only clerks so they can only talk about screens and reports."

As a consequence of using ToolKit some of the designs were not always appropriate, as noted by a project manager:

> "Developers tend to see things only through the tools, so they don't think through the functions. And the problem is that we start designing from screens and reports, not the functions of the business. And we have no real sense of what people do and what they need in the business areas. So the inputs govern what the system does, and the tools don't address the functional thought process. That's ridiculous. We've found a whole bunch of screens and reports that are absurd and quite useless, that don't serve any useful function."

While the above points demonstrate the bounding effect that design methodologies and CASE technologies have on the process and outcome of design, a further consequence is impact on the developer. A developer made to think and converse in a restricted language will be unable to formulate solutions that extend beyond the structures available in the language. The following quotes by two analysts reflects this consequence:

> "Tools force people to think in a certain way. We all think screens and reports. So we don't have a chance to think if things could be done a better way. ... Tools have definitely stopped me thinking about other ways of doing things. I am not thinking myself because the tool does it all for me. We bring a single mindset to the different projects, and so we already know what to do."

> "When you rely on tools you inherently assume certain things, and hence this hinders your ability to see other things. To make an analogy, it's like playing with a pack of cards: you have to pick a card out of the 52 available; you can't pick the 53rd. So tools create a structure to work with, but we fall into the trap of not seeing beyond it."

In these cases the CASE tool has so successfully facilitated a standardized way of executing systems development tasks, that the mere interaction with ToolKit has come to define the work of systems development. Systems development work is seen less as active engagement in problem-solving, and more as abstracted symbol manipulation. This perception was particularly

noticeable among the more junior developers on the projects, who were largely unable to discern the range of possible design directions that they had implicitly excluded from consideration because of ToolKit's standardized approach. Indeed for the majority of the very recent recruits who had little or no other systems experience, systems development was understood as following the directions of ToolKit. That other ways of developing systems were possible was dimly perceived, and the semantic bounding was poorly understood.

This discussion reflects the inevitable coupling of methodology and activity, that is, how a given activity quickly becomes so coupled to the methodology that guides it, that it becomes difficult and inefficient to conceive of alternative ways of doing it. A senior analyst commented:

> "By using the tools we are reinforcing the methodology. But at the time of using the tools we aren't exactly aware of how the methodology underlies the tools. So we're hiding the methodology in the tools, and the use of tools forces our use of the methodology and promotes certain work habits without our awareness."

Standardizing on a single design perspective leads to that perspective becoming taken-for-granted, so that the semantic bounding effects of a CASE technology are institutionalized as developers internalize that bounding, and make it implicit. Where action is implicit, reflection in action is constrained, and we should expect to see less questioning of the underlying assumptions. Lack of awareness or reflection constrains not only the outcomes of systems development, but bounds developers' problem-solving endeavors. They do not exercise alternative design strategies, or contemplate different views of the problem. A senior project manager noted:

> "It seems with tools it's easier to shield nonreflectiveness among developers. They can hide behind the tools which they couldn't really do before. Thus developers using the restricted design vocabulary of a CASE technology may not develop the design acumen or perspective that leads to creative work."

In general throughout the projects investigated, the interaction of the analysts with ToolKit was passive, with the tools being used to record information such as texts, diagrams, and data descriptions, and to generate interface designs, dialog simulations, or program code. A senior project manager expressed reservations about the long-term viability of blindly adhering to CASE technology, which he termed:"the cookie cutter approach to systems development," acknowledging that:

> "Design tools are not real replacements for designers, they are aids. But people start using them as substitutes for thought. We try and leverage off our tools, so we tend to use staff at lower levels. But these people think that if the reports and screens are designed then the design is done. But they may not have thought things through properly. They tend to get infatuated with tools and

> lose sight of what the task is supposed to be. People tend to get so wrapped up in the tool they forget to think. So we get nice pretty screens and reports, but we don't get better quality designs."

This manager's insight was confirmed by numerous observations that ToolKit appeared to encourage developers to focus on the form of development work rather than on its content. Within our current study such behavioral pattern will not be considered bounding effects, as although they are obviously very real, they are not inherent in CASE technology, but rather reflect other effects of the interaction between users and their information technology. This tendency of CASE technology to mediate a particular interaction between developers and tools, leads us closer to the syntactic bounding that such technology exhibits. The following subsection examines some of the ways in which ToolKit restricted design work at the level of syntax.

3.2. Syntactic Bounding Effects of ToolKit

The experience with ToolKit highlights the role of implemented design aids as a surprising source of some rather annoying bounding effects. As an example, ToolKit's screen design aid suggests "ergonomic" screen designs in response to input of data items and their characteristics. Such an algorithm relieves developers from tedious layout of screens and attention to interface standards, as well as dramatically improving the productivity of interface design.

However a number of unanticipated consequences have emerged from use of this design aid. First, routine screens are trivially simple to design, while nonstandard screens - which are not accommodated in the screen design aid - appear in contrast, to be complicated and time-consuming. In effect this restriction in the ToolKit syntax (inability to support nonstandard screens) has made exceptions problematic, encouraging extensive avoidance behavior on the part of the analysts. Analysts reported and were observed attempting to dissuade users from demanding unusual screen designs that would require them to design screens outside of ToolKit, and hence incur penalties of tedium and lost productivity. If this failed, analysts would manipulate the ToolKit algorithm by juggling parameters in an attempt to force it to produce different designs. A project manager commenting on this tendency, noted:

> "The standards work great for simple screens, but for complex screens they're no good. So analysts spend two days trying to fool the tool to get it to do what they want. Part of the problem is that tools assume their standards are perfect development standards for the whole world, and you can't adjust them to do what you need to. The rigid standards in the tools may not always best suit the environment of a particular project."

The ToolKit experience raises interesting question about the role of integration with respect

to syntactic bounding effects. Although the lack of integration introduces a "foreign" concern into the process, a higher level of integration potentially tightens the tools control over the order in which tasks are performed. For instance, ToolKit prohibits execution of tasks, unless the defined preceding work meets the tools' completeness criteria. This ensures that production tasks are executed in the prescribed order, and that all the relevant information is available before a task is attempted. ToolKit further monitors the execution of tasks by doing extensive cross checking of the design documentation, determining inconsistencies and ambiguities, warning analysts when errors or omissions are detected. However it does not always make sense to perform tasks in a strict sequential order. In particular the nature of design is such that it is a highly iterative process. Iterative development (prototyping) for example, encourages constant feedback and backtracking to accompany development.

A related source of syntactic bounding effects is the way support for design teamwork is implemented. On a multi-person project, members are often waiting for colleagues to complete tasks whose output they require. Analysts on the projects investigated noted that they would often try to trick the tools by creating the appearance that a task had been completed, so that the tools would let them get on with some other work.

In the section on semantic bounding we illustrated how methodological restriction had encouraged a passive mode of interaction between ToolKit and developers. Such passive interaction is accentuated by the specific manner in which ToolKit's capabilities have been implemented. For example, the data structure editor in ToolKit, while it supports multiple hierarchic levels, is not well suited to the size of the screens being used to represent the data structure designs. The text always appears too small to be easily read, so that the multiple levels of logic are difficult to understand by merely glancing at the monitor. The lack of flexible cursor control and the small screens make easy manipulation of the data structure elements tedious and complex. As a result attempting to derive the data structures at the machine interface proved too onerous, and most of the developers first generated the data structure designs by hand with paper and pencil, and then transcribed the completed designs into the data structure editor, using this editor merely as a documentation capability. Such technical limitations -- often arbitrary -- can have a significant influence on the design process.

Syntactic bounding effects may have their own derivative behaviors, echoing similar

comments on the behavioral patterns encouraged by ToolKit features that are closely related to sources of semantic bounding effects. For instance, ToolKit supports increased rework, as editors facilitate ease of modification. A senior analyst pointed out that:

> "We can do things faster and easier, so we can keep coming back to refine the work. So like writing papers on a word processor, the tools allow iterative refinement of the designs."

But ease of refinement has unanticipated consequences, as an analyst noted:

> "We have found that as with all things, the professionals have started to get carried away, doing too much work and generating more documentation. People have a tendency to fill out forms because they're there. So we don't get the productivity savings we should."

Since this is again an adapted behavior, rather than behavior dictated by the way the CASE tool is implemented, we do not consider this set of phenomena within our framework.

These excerpts from an empirical study of a specific CASE technology highlight some characteristics of CASE systems that gave rise to bounding effects in the analysis and design process. Clearly such effects are "real," having material consequences not only for the nature of the design process and the behaviors of designers/tool users, but also for the ultimate information system that is produced, and its users. The findings emerging out of this study led us to speculate about the nature and extent of bounding dimensions in other CASE offerings available on the market today. The following section reports on a survey of current CASE technologies which we examined from the perspective of our bounding effects framework.

4. A Survey of Bounding Features of CASE Technology

The notion of comprehensive assesment of semantic and syntactic bounding in CASE technology is yet largely undefined. The critical review in this section adopts the bounding aspects highlighted in Section 3 above as a working definition, and focuses on the corresponding bounding features, as further elaborated in the respective sub-sections below. The second working assumption here is that the term "CASE technology" can be operationalized as the CASE products that have a practically measurable share of the installed CASE base. Throughout this section we therefore summarize potentially bounding features across products, without regard to their identity (with very few exceptions where such generalization would be too strained). In Section 5 below we take issue with this approach, and suggest some alternative courses for future research.

This review is based on Chris Gane's recent survey of the CASE market [4], which

identifies about 30 CASE product offerings. Figure 4-1 lists the 22 products which for all practical purposes define the installed CASE base. Products are listed in decreasing order of their estimated share of the installed base, along with estimates of the pace of shipment of new systems (based on the data in [4]). In our analysis we have distinguished between what we label the major CASE products, the top seven products that account for almost 85% of the installed CASE base, and the remaining products. Gane's report provides a uniformly structured, concise technical description of 25 products, based on vendors' material. The description is meant to highlight the nature of each of the CASE products with respect to a number of dimensions, e.g., diagrams types supported, integration between the graphic interface and the repository, project management services and the like. The 25 product descriptions included in the report are used here as raw data, from which we draw some general observations about potential semantic and syntactic bounding effects of CASE products.

4.1. Semantic Bounding in CASE Products

From the vantage point of semantic bounding effects, Section 3 pointed to the extent to which the vocabulary of analysis and design is restricted by the CASE product, and the extent to which the process of design is influenced or shaped by it. It is interesting to note that although the inevitable constitutional and methodological effects on the design ***process*** featured so prominently in designers' commentary and experience as reported in Section 3, these effects are rarely addressed elsewhere, and in particular are not dealt with at all in the Gane's survey. Although every CASE tool necessarily embodies -- and enforces -- some model of the analysis and design process, our collective awareness of this subtle impact is indeed still in its infancy.

References to the design process are made only with respect to the very few CASE products that relate to it explicitly. Knowledgeware Inc's Information Engineering Workbench (IEW), is the more popular of these products (see Figure 4-1). IEW recognizes three stages in IS development, namely planning, analysis, and design, with a relatively high level of integration among the three corresponding modules. Integration is achieved primarily through an "Encyclopedia" which is managed by an Expert System. At the time of the report IEW centered around Entity-Relationship Diagrams (ERD) and did not support any other system development methodologies, nor did it offer the flexibility to enter the development lifecycle at any point desired by the user.

Product	Shipments 1988	Installed Base 1988	% of Installed	Growth in 1988
EXCELERATOR	8500	17500	28.89%	94.44%
IEW	4000	9600	15.85%	71.43%
VAW	4000	6300	10.40%	173.91%
DesignAid	3000	6000	9.90%	100.00%
TEAMWORK	2000	4000	6.60%	100.00%
AUTOMATE	2000	4000	6.60%	100.00%
AD/T	2000	3500	5.78%	133.33%
vsDesigner	1000	1550	2.56%	181.82%
SW/Pict	900	1400	2.31%	180.00%
ProKitWB	1000	1400	2.31%	250.00%
ANATOOL	600	1000	1.65%	150.00%
DEVELOPER	700	1000	1.65%	233.33%
ERD	600	900	1.49%	200.00%
Deft	250	550	0.91%	83.33%
Meta Tools	150	450	0.74%	50.00%
MAESTRO	80	360	0.59%	28.57%
TELON	90	350	0.58%	34.62%
CorVision	150	300	0.50%	100.00%
APS	85	215	0.35%	65.38%
IEF	50	130	0.21%	62.50%
TRANSFORM	20	60	0.10%	50.00%
MULTI/CAM	0	14	0.02%	0.00%
Total	31175	60579	100.00%	106.02%

Figure 4-1: CASE Products and Their Installed-Base Share and Growth

In the assessment of CASE design vocabulary, the two primary aspects suggested by Section 3 as indicators of potential semantic bounding effects are the lack of a variety of "design objects" (our label for the collection of diagram types and repository objects), as well as the extent to which idiosyncratic design objects are used. These two aspects practically define the content of the analysis and design activities. The operational interpretation of these aspects guided the analysis of the population of CASE tools with respect to semantically bounding features. Specifically we focused on (1) the number of design objects that each tool offered, (2) the variety of design object types, and (3) the rarity of design object (i.e., design object types which are more unique in the sense that they are offered by one or two CASE tools only). In the following paragraphs we refer to each of the above aspects in turn.

It seems that the major CASE products offer a fairly broad variety of design objects (See Figure 4-2) -- offering between 4 and 13 diagram types, with mean of 8. When the major CASE products are tallied with respect to their repository objects, the variance is quite unsettling -- Index Technology's Excelerator, the single most popular CASE product offers no less than 32 repository objects, while Visual Software's Visible Analyst Workbench (VAW) offers just three. It should be noted that one of the objects offered by VAW is "text" which does imply a rather flexible repository. Two other products among the major seven tools do not define separate repository objects, but rather adopt diagram concepts, refered to as "diagram objects," as their underlying repository schema. As far as bounding effects are concerned, the more elaborate schemata probably impose more specific bounds, as the granularity of the design data is finer and therefore more specific and confining.

	Number of Diagrams	Number of Products	Number of Repos. Obj.	Number of Products
	4	1		
Major	7	2	3	1
CASE	8	3	11-13	3
Tools	13	1	32	1
			Diag.Obj.	2
	0	1	4-11	9
	1	4	12-29	3
Other	2-4	6	Diag.Obj.	3
CASE	6	3		
Tools	7	1		
	9	1		
	n/a	2	n/a	3

Figure 4-2: Distribution of Design Objects Offered in CASE Tools

CASE tools collectively offer a broad set of commonly defined digramming methodologies. The seven major CASE tools offer, as expected, a selection of commonly defined diagrams types like Data Flow Diagrams (DFD), Entity-Relationship Diagrams (ERD), Structure (or Decomposition) Charts, Flowcharts or Process Flow Diagram, State Transition Diagrams, Warnier-Orr Diagrams, Action Diagrams, Jackson Diagrams, Bachman Charts, DB2/Table Charts, ADABAS File Charts, and Decision Tables. The remaining tools further support Nassi-

Shneiderman Diagrams and HIPO Charts. Bounding features are therefore to be expected not in the general variety of diagramming options but in the specific options selected for a particular CASE tool. A further mitigating factor in this respect are facilities for customizing diagram types. As it turns out five out of the seven major CASE products offer various facilities to customize diagram types or diagram elements, while only five of the remaining products offer comparable customizing facilities.

Based on this rudimentary analysis it seems that bounding features are not apparent in the range of diagrams. However, six out of the seven major products included "rare" diagram types that basically constitute a specialized tool vocabulary. The number of specialized diagram types per tool ranges from 2 to 5, with mean of 2 to 3 diagram types. Of the remaining 18 CASE tools analyzed, nine included only commonly defined diagrams and the rest included between 1 and 4 specialized ones. In general, it seems that the more a tool supports design activities, the more it tends to offer specialized diagram types.

The likely effect of rarity is the introduction of "semantic influence" into the process which is uniquely due to CASE use, and which is **not** inherited from widely exercised methodologies. Sixteen such specialized diagrams were introduced by the major products alone. Examples include Document Graphs, Booch Diagrams, Visual Real-Time Diagrams, Free-Form Graphics, Entity Life History, On-Line Dialog Diagrams, Transaction Dialog Diagrams, Batch Run Flows, and Module Sequencing Charts. Fifteen additional specialized diagrams were introduced by the remaining CASE products. They include Operation Procedures, Entity Hierarchy Diagrams, Process Dependency Diagrams, Module Networks, and Function Networks, among others. The actual semantic bounding effects of these idiosyncratic diagram types were illustrated in Section 3 above.

Turning now to the repository, we note that it probably more than anything else, reveals the semantic richness of a CASE tool, exposes its underlying conceptual basis, and serves as a fundamental source for potential bounding effects. It is therefore interesting to note that besides the expected, commonly defined repository objects, practically every product on the market includes at least one object which is not shared by any other product. The list of these commonly defined repository objects (typically shared by two or more CASE tools) include:

Diagram objects, problem/requirement or requirement, external agent or entity, event, state,

entity and entity type, relationship and relationship type, record, data structure, data element or item or field, attribute type, data store, database/file, data flow, process, and module.

The ratio of idiosyncratic objects supported by a CASE tool to the number of design objects in the entire repository vocabulary varies widely, from as low as 16% (one out of six), and up to 100% (four out of four). The list of those idiosyncratic objects is surprizingly long, but its significance to the discussion of bounding effects cannot be overstated -- it seems that every CASE product introduces a different dialect, or at least a special set of nouns into the language of systems analysis and design. Examples include:

22 different relation types, subject area, users, global attribute, application entity, cluster, panel and screen definitions, screen system function, access modules, error handling narrative, exchange, functional primitive, condition, set subtype, group, input, interface-memo, output, system parameter, unit, configuration, basic data, known data, glossary item, text, and miscellaneous.

The meaning of quite a few of these concepts is not self-evident, but for our discussion it suffices to note that these terms exist, and that they are an integral part of the respective CASE products. Indeed, some of them may be substantively similar to more common concepts, however the fact that their respective vendors have elected to rename them suggests that some differentiation might have been attempted, which in turn introduces a potential bounding effect. It also means that every user of a particular CASE tool will have to become conversant with the semantic behind these idiosyncratic concepts.

4.2. Syntactic Bounding in CASE Products

From the implementation perspective we are interested in arbitrary bounding features that are part of the delivery platform, such as the extent of integration within the various parts of the implemented CASE tool and its multiuser environment. Such syntactical bounding aspects represent instances where the designer has to shift her attention from the task of system analysis and design, to overcoming some "irrelevant," product specific peculiarities in data entry, display, storage or processing. This section parallels Section 3 by examining four syntactic bounding features, namely technical limitations, the extent of design assistance, the degree of integration, and the support for multiple users.

Technical limitations of CASE tools reflect tool implementors' choices and are often arbitrary in nature. Among the seven major products we find the following limitations:

75 objects per diagram, 300-500 "boxes" per diagram (implied by a 64K storage limitation per

diagram), diagrams made of text symbols only, charts that cannot spread beyond the width of 120 characters, and only 6 levels of DFD explosion.

Some of the remaining products list the following technical limitations:

Diagram size that cannot exceed 11"x15", 36"x48" or "3 by 3 screens," no more than 800 symbols per diagram, 200 symbols in another product, no more than 15 processes per diagram, only 4 levels of DFD explosion, or 9 levels of explosion in another case, no more than 300 objects with an average of 10 attributes each, no more than 128 attributes per object, and no more than 100 datasets.

Not all products were marked with limitations of the above nature. Does that mean that they are syntactically flexible? Probably not. As the notion of bounding effects amply clarifies, limitations and bounding effects in information systems -- which CASE products ultimately are -- remain mostly hidden, and are rarely realized in advance. They typically reflect what the designers of the CASE tools thought were "sufficient" capacity and capabilities at the time of development.

It appears that the extent of design assistance embedded in a CASE tool introduces a potential syntactic bounding effect. Extensive assistance implies a stricter adherance to standard expressions, and less opportunity for the designer/user to exercise control over the process. In a typical design process requirements and specifications are disambiguated in a gradual fashion, but automated design assistance usually requires an early and complete specification of the details upon which to base its conclusions. Although the trend is toward more such assistance in CASE products, design assistance is not a common characteristic of current CASE technology. Of the seven major CASE products only three include such a facility, focusing on diagram consistency, affinity/similarity analysis, design tools for relational and hierarchical databases, as well as flat files. Of the remaining 18 products ten do not offer any such facility, while the others provide services like Normal Form analysis (especially 3NF), advice on relational data models, support for physical database and system design, tracking (and recommending) progress in the logical development process, consistency checking, and suggestions on modular design.

Design assistance becomes the distinguishing aspect in some of the more recent sophisticated tools, which typically employ Artificial Intelligence resources for extensive design support (e.g., expert system for performance-optimization). In some other cases companion packages (like Mini-Asyst) are offered as "add-ons." However, few vendors have recently announced specific intentions with repect to design assistance.

A characteristic bounding effect in the use of CASE tools stems from the degree of

integration -- or the lack of it -- among the various parts of the tool. Manual integration implies artificial break points in an otherwise continuous process, break points where data from, say, the diagramming stage, is explicitly moved forward to the next stage, e.g., a more detailed analysis or actual design. Lack of integration has a clear bounding potential in a number of ways, key among which are (1) the partitioning of the gradual process of analysis and design into a more discrete, rigidly staged one, and (2) the subtle effect of a constantly forward moving process. The latter is reflected in the notion that "integration" in a CASE tool is the automatic ability to create repository objects from diagram elements. Nevertheless, the reverse direction is as important, and some of the products start to address this issue as well. Too tight an integration introduces another bounding effect, which enforces some rigid translation of, say, design data, into the next stage, namely actual code. For example, the automatic transformation from design guidelines to a set of program shells clearly bounds the likely path a system structure can take.

The survey which is the basis of this analysis focused on the diagram-repository linkage as a measure of integration. Of the seven major CASE products three provide only manual procedures of getting diagram information into the repository. The other four provide differnt levels of integration, from "immediate automatic" to "automatic update of repository objects upon validaticn of diagram." The remaining 18 products offer similar a variety of modes of integration. For example, four products offer no integration or limited manual procedures, two products update the repository in response to request to SAVE a diagram, and ten offer automated or "fully" automated integration with one or two way consistency maintenance between diagrams and repository objects, and real-time update of repository information. It seems then that the bounding threat of CASE technology lies more with over-integration and the appropriation of control from the hands of the designer/user.

The last of the syntactic bounding features to be considered here is the degree of support for team work in the design process. Design is a communication process, and a CASE tool that recognizes this allows its users to interact more naturally, without demanding "irrelevant" attention to the underlying mechanics of repository maintenance. Multiuser support can be achieved in multiple ways and multiple degrees. As exemplified by the major CASE products -- six of which address multiuser work -- such support can be achieved by various degrees of "locking" as well as mechanisms for merging multiple copies into a consistent central repository. One product

provides a"check-out" mechanism, which locks the entire database, and thereby enforces single-user update. Two other products lock repository information at a finer granularity, (e.g., records or repository "parts"). Another product does not lock, but rather generates multiuser warnings about diagram inconsistencies due to concurrent update. Finally, one CASE tool assumes distributed work and consistently consolidates workstation versions into the central repository.

The remaining products basically replicate the range of multiuser support of the major products. Specifically, some offer no support at all (7 tools), others define sequentialized single-users or limit access to the data's owner (2 tools), and locking is provided at various levels, e.g., diagrams, sub-diagram, documents, objects, application, or release, with corresponding check-out protocols (7 tools). One product offers locking services through the DBMS with which it is implemented (DL/1 locking), and two others either generate warning messages about possible inconsistencies, or manage temporally synchronized access to repository versions, using time-stamping mechanisms.

Throughout this section we have highlighted various CASE characteristics which could give rise to bounding effects in the process of systems analysis and design. Section 3 argued that these "threats" do materialize, and that they are "real." In the next section we turn to review the implications of these observed effects for the various constituencies of the IS community.

5. Implications

In this paper we have gradually refined a framework for examining the nature and impact of bounding effects in CASE technology. Refinement was initially conducted through the study of the revealed effects in a situation of actual use of a CASE tool, and further in the examination of a recent snapshot of CASE technology. Not only have we demonstrated that bounding effects have real consequences (section 3), but also that a surface examination of CASE tools can reveal potentially bounding features (section 4).

Bounding, in itself, is not "good" or "bad", but rather needs to be recognized for what it is. Bounding is the premise upon which design methodologies are established: "our entire ability to attend presupposes our experiencing such discontinuities between what we focus on and what we perceptually ignore" (p.3 in [10]). The main cognitive function of boundaries is to provide the basis for sorting out complex phenomena, a means to "separate supposedly discrete chunks of

realities from one another" (p.23 in [10]), or, to paraphrase Russel Ackoff, the mechanism with which we formulate the mess. The centrality of bounding in IS design is emphasized when one notes that the word "definition" is actually derived from the Latin word for "boundary" (*finis*). Requirement *definition* in the various levels -- conceptual, logical and physical -- is the core of system engineering.

In this section we review our findings and consider their implications for information systems practice, education and research.

5.1. Implications for IS Practice

It seems that the most urgent implication for IS practitioners with respect to bounding effects is that of awareness of their existence. The ultimate understanding of bounding effect may well take the form of a contingency framework, as the extent to which a CASE tool may prove constraining depends on how appropriate the supported design perspective is to the problem at hand. Where the restricted design vocabulary available through a CASE technology adequately captures the phenomena to be modeled, the bounding effect is enabling, that is, it provides designers with a structure within which to focus on pertinent aspects of the problem, resulting in a suitable design resolution of the problem. The ability to avoid too much attention to detail allows users of CASE technology to be much more productive in certain stages of the development life cycle.

However to the extent that the restricted design vocabulary is not adequate for modeling a given problem, enforced use of the vocabulary will unnecessarily constrain the design solutions generated. The restriction of the range of design objects is bounding as users' design solutions are limited to those designs that can be conceptualized, articulated, and deliberated within the restricted design vocabulary available in the supported design approach. CASE tools typically cannot capture doodles, pictures, photographs, metaphors, jokes, idiograms, or audile signals. When interviewing a user to determine information requirements, how does an analyst record body language? Inevitably the real world information represented via the medium of CASE technology is restricted, rationalized, standardized, and made consistent. As a consequence of bounding developers often ignore -- sometimes deliberately, usually inadvertently -- many realms of organizational life (like conflict, contradiction and irrationality) that are not expressible in the

vocabulary of the tool they are using. And it is on the basis of this sort of sanitized information that systems are built.

The process of information systems design can be viewed from different perspectives. One recent perspective, suggested by [1], emphasizes the affective dimension of designing, namely that the design activity is a process through which designers become comfortable with a set of specifications. This applies in particular to the early stages of the IS design cycle, i.e., conceptual design is so removed from any direct measurement of the quality of the outcome that it must primarily rely on whether the designers are indeed "happy" with the blue print they have come up with. Such a view of the design process emphasizes the role of design methodologies in facilitating the convergence of the design team or individual on an agreeable design. CASE technology, indeed as an enforcer of methodological structure, redefines the meaning of what it is to be "emotionally comfortable" with a proposed analysis or design. Attention will typically shift from the designers' sense of comfort to the seemingly concrete, rather visible "mechanic comfort" expressed by the CASE tool.

The discussion of the advantages and disadvantages of CASE thechnology has to take place within the broader framework espoused in this paper. We have postulated three levels of bounding that CASE technologies bring to the design process: constitutional, methodological, and implementation bounding. These three levels of bounding are engendered by various semantic and syntactic features of the design process. Practitioners should carefully consider the implications of acquiring a CASE tool in the light of the potential bounding effects that such technology will introduce into their workspace. As we have tried to elucidate, bounding effects are potentially enabling as well as constraining. A careful assessment of the organizational context and conditions under which a given CASE tool will be used, together with a thorough understanding of the tool's features, can provide much insight into determining its potential bounding impact. Any consideration of the advantages and disadvantages of CASE technology inherits the constitutional and methodological properties of its underlying design philosophy, and amplifies these through particular implementation details. Some of the advantages and disadvantages have been generally stated in Figure 5-1.

Bounding Level	ADVANTAGES	DISADVANTAGES
CONSTITUTIONAL	Expedites design process Facilitates convergence Guarantees consistency Provides quality criteria	Imposes presuppositions about the task and its nature Assumes away or ignores some aspects of the task
METHODOLOGICAL	Eliminates search for the "appropriate" approach	Not every approach can achieve the same level of effectiveness in all situations
IMPLEMENTATION	Provides ready-made templates through which design activities can occur	Does not accommodate the "grey" areas of design Forcing resolution even where inappropriate

Figure 5-1: The Bounding Effects of CASE Tools

5.2. Implications for IS Education

This study ties into the ongoing deliberation in many schools about the nature of the systems analysis and design course. In particular, what is the role and place of CASE tools in the IS curriculum? On the one hand CASE technology is an obvious ingredient in contemporary information systems literacy, while on the other hand CASE tools are supposedly mere tools, namely they do not represent intellectual content that is substantively different from the established principles of IS analysis and design. The concrete question is therefore whether to adopt a CASE tool as a pedagogical vehicle in the systems analysis and design course.

Our findings imply that such adoption may not be very appropriate, on a number of accounts. A relevant finding is the observation about novices and CASE tools, and the misperceptions about systems analysis and design that tools foster in inexperienced users.

Junior anlysts equated tools with substantive design content, and avoided developing a deeper notion of the plurality of problems and the variety of activities so characteristic of system analysis and design. If a CASE tool is to be used throughout the IS curriculum, a danger may be that it will promote the trivialization of the system analysis and design skills.

Further, anecdotal evidence suggests that courses that adopted a CASE tool reduced to frustrating exercises in overcoming the syntactic bounding effects of the tool. A relatively large proportion of class time was spent "struggling" with the specific product, emphasizing technical matters of expression, and neglecting more conceptual tenets of system analysis and design.

In spite of all these caveats, the value of CASE technology in IS education should not be overlooked. A CASE tool can provide the necessary concrete system analysis and design experience upon which inexperienced analysts can start to build more elaborate concepts. In parallel to the discussion of the value of a CASE tool to the practitioner, a CASE tool in a school environment may allow students to experience a realistic system analysis project, while without it they may have to limit their application to either solving "toy problems" or do only partial analysis. An introduction of CASE technology might be effectively achieved in an advanced system analysis and design course.

5.3. Implications for IS Research

The study has raised some interesting further questions. In light of the findings reported here a deeper investigation into the actual use of the many CASE tools is indeed warranted. Such an investigation should attempt to determine the actual workings out of bounding effects in practice, and to assess the meaning of the various levels of bounding to designers in the conduct of their work.

One specific way to approach the study of semantic bounding effects is to examine the richness of a CASE tool vocabulary. To do this we can adopt a language system perspective. Specifically, if we think of design objects as constituting the vocabulary of design then we can try to assess how they facilitate the articulation and conceptualization of complex worlds. Daft and Wigington [3] argue that language systems vary in the variety they contain for matching environmental variety. Low variety in a vocabulary thus may produce poor problem-solving behavior in complex environments because most of the environment is not sensed or "not seen"

[9]. By positing CASE technologies as language systems, we can then examine the variety they support by determining the extent to which restriction of variety bounds the problem space which CASE users can meaningfully enact with the tools.

Closely related to variety is the rarity or idiosyncracity of a CASE tool's vocabulary. To the extent that CASE technology introduces new concepts and objects -- and there is ample evidence of this in Section 4 -- it creates its own dialect, forcing users to adopt an uncommon form of design expression. The extent to which such syntactic conformity restricts or enables design expressability was not examined in our study, and is an important area of future research. In section 4 above, we speculated that such idiosyncracy was likely to increase the bounding effects of a tool over one that implemented more widely recognized design objects. However this latter kind of tool can lead to further bounding as well, where the common objects, because they are so widely used, easily slip into nonreflective subconciousness. Hidden from view, such taken-for-granted concepts exert a subtle, yet powerful influence on designer problem-solving behavior. When such bounding is enabling and when it is restrictive clearly bears further investigation.

A working assumption adopted in Section 4 was that the term "CASE technology" can be operationalized as the collection of CASE tools, and that observations about the state of the technology can be made by examining this aggregate. While this seems adequate as a "first cut" and fits the preliminary nature of the study of CASE technology in general, the approach has obvious shortcomings. Prominent among them is the failure to discern interactions among the different bounding features, a deeper analysis of each of the features, as well as a closer look at, and a more universal classification of, design objects. This entails the development of a multi-dimensional classification scheme of CASE tools, with respect to the recognized bounding features. A useful starting point may be the functional model of IS planning and design support technology proposed by Henderson and Cooprider [6].

Another aspect of CASE is that it is computerized support for the process of system design, and therefore has -- potentially -- side effects similar to the effects managerial information systems (e.g., DSS) commonly have on the tasks they purportedly support. Silver [8] has examined a feature of DSS he terms "restrictiveness," which seems closely related to the concept of bounding. An examination of the similarity and dissimilarity between the two concepts could broaden the understanding and significance of both.

6. Conclusions

The media through which the design process is executed -- methodologies, languages, aids, and CASE tools, apparently do impact the message. To large extent this is intended -- the role of these design media in improving the quality of the resulting information system has always been a cornerstone of IS practice, education and research. Questions, if raised, dealt with how well they function, or whether one approach is indeed better than another. Bounding effects, however, have remained largely implicit. "Nothing evades our attention so persistently as that which is taken for granted" noted Gustav Ichheiser, which might explain why this rather obvious perspective of IS design has not been more widely or rigorously addressed so far. The study of bounding effects indeed represents a complex change between "figure" and "ground" in the study of IS design, and introduces new opportunities for better understanding of this domain.

References

[1] Ariav, G. and Calloway, L.J.
An Examination of the Use of Dialog Charts In Specifying Conceptual Models of Dialogs.
Technical Report, Information Systems Department, New York University, July, 1988.

[2] Carlyle, R.E.
Where Methodology Falls Short.
Datamation 34(24):179-191, December, 1988.

[3] Daft, R.L. and Wiginton, J.C.
Language and Organization.
Academy of Management Review 4:179-191, 1979.

[4] Gane, C.
Computer Aided Software Engineering: The Methodologies, The Products, The Future.
Technical Report, Rapid System Development, Inc., 211 West 56th St., New York, N.Y. 10019, 1988.

[5] Giddens, A.
New Rules of Sociological Method.
Basic Books, Inc., New York, NY, 1976.

[6] Henderson, J.C. and Cooprider, J.G.,
Dimensions of I/S Planning and Design Technology.
Technical Report 181, CISR, Massachusetts Institute of Technology, September, 1988.

[7] Orlikowski, W.J.
Information Technology and Post-Industrial Organizations: An Examination of the Computer-Meditation of Production Work.
PhD thesis, Stern School of Business, New York University, 1988.
(Unpublished).

[8] Silver, M.S.
On The Restrictiveness of Decision Support Systems.
Technical Report 4-88, Information Systems Research Program, UCLA, November, 1987.

[9] Weick, K.E.
Organizational Communication: Towards a Research Agenda.
Communication and Organizations: An Interpretive Approach.
Sage Publications, Beverly Hills, CA., 1983, pages 7-29.

[10] Zerubavel, E.
The Fine Line: Boundaries and the Classification of Social Realities.
The Free Press, New York, NY, 1989.
(Draft, in Preparation).