Implementing tool support for box structures

by B. S. Tagg

This paper describes a feasibility study to implement partial tool support for the graphical component of the box structure methodology (BSM). By following the defined strategy and process, an existing computer-aided software engineering (CASE) environment has been extended with a customizer to provide support for the box definition graphics (BDG) component of BSM. The critical functions required from a CASE environment are also described to provide the reader with a background for selecting one of the various implementations available today.

The creation of the box structure methodology (BSM) provides systems developers with a new powerful, yet straightforward, software engineering methodology.[1]

After the inception of BSM, a course of study was developed to educate systems developers in the creation of requirements specifications[2] by employing BSM to solve problems involving the design of systems. The expectation was that as more developers became trained in its use, BSM could evolve into one of the mainstream methodologies used to describe and define systems.

Since BSM is relatively new, it has had to compete against other, more established methodologies such as structured analysis and structured design. Sophisticated support environments were developed to provide functions for creating and analyzing the deliverables of these methodologies. When BSM was introduced, it lacked a support environment that would provide functions to enter and analyze BSM data. Without such functions, the strategy for incorporating BSM in the internal development process was weakened.

This paper describes the study developed to create a support environment for BSM. The first section describes a strategy developed to implement computer-aided tool support by using a computer-aided software engineering (CASE) tool customizer. The conceptual view of a CASE environment is introduced along with the view of an extended environment representing the new customized functions created. With a strategy identified, the next section defines a process that results in the successful integration of functions supporting the entry and analysis of box definition graphics (BDG). The process begins with learning the methodology from a user's point of view, and then describes the methodology with an entity-relationship diagram so that its components can be determined. A target CASE environment is chosen, learned, and implemented. Successful results are achieved by following the strategy and procedures explained in the section on BSM tool results. Finally, the validation process is presented.

The major product of this study was the successful implementation of limited BSM support. The support

Figure 1 Logical structure of existing CASE environment

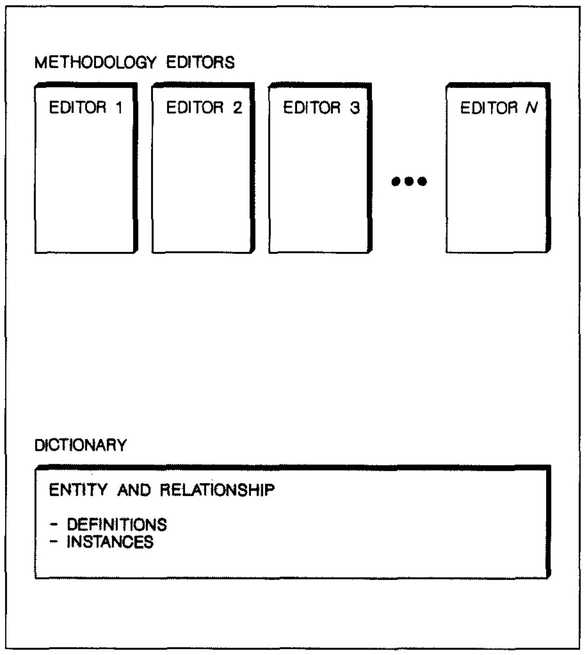
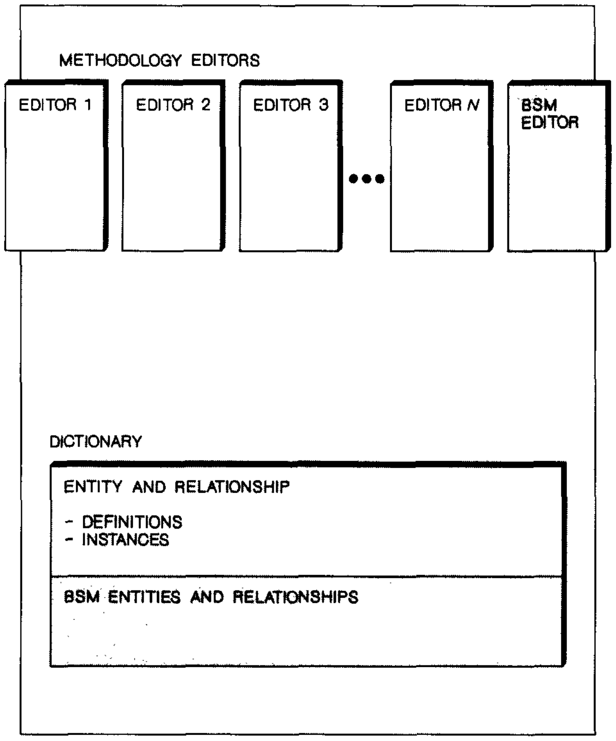


Figure 2 Extended CASE environment



is significant, since it represents a pioneering effort in providing BSM users with an environment that allows the creation of BDG in an existing, integrated CASE environment.

BSM tool support strategy

Numerous papers have been written containing the evaluation of available CASE technology.[3–6] The majority of these papers deal with the methodologies supported by current CASE environments and how these methodologies can be applied to design software systems. Few, if any, of these papers deal with the aspect of customizing these environments to support new software engineering methodologies.

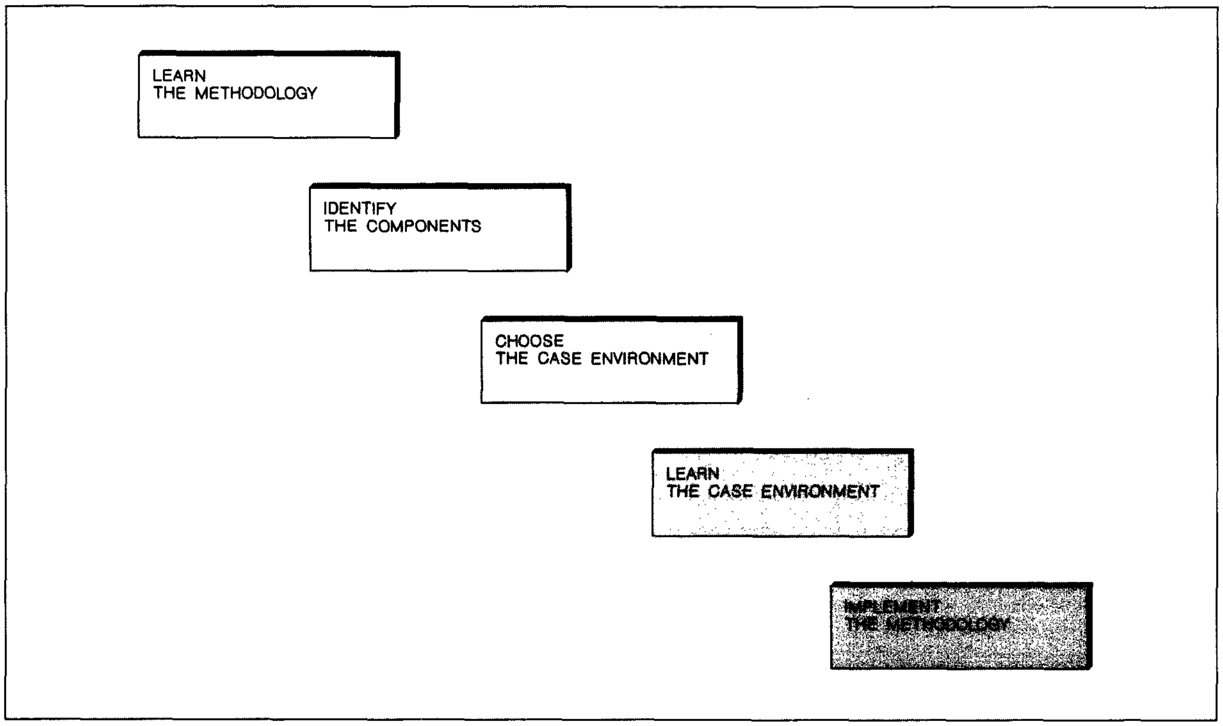
The focus on CASE tool customization has emerged recently with the creation and increased popularity of BSM. Proponents of this software engineering methodology were limited in its use because there was no tool support. To address this problem, a strategy was defined that would lead to the creation of the required computer-aided tool support.

In developing this tool support, the tool builder is immediately faced with a choice between creating a stand-alone tool or extending an existing CASE environment. Since a critical requirement in providing tool support for BSM was to have BSM integrated with other, more mature software engineering methodologies, the strategy defined here is based on extending an existing environment that already supported the more common methodologies and diagramming techniques such as data flow diagrams, structure charts, and entity-relationship diagrams. Several existing CASE environments meet this criteria and provide customization support. Customizer® from Index Technology Corporation and SYLVA™ Foundry from Cadware, Inc. are two such environments that were both used to validate the strategy and process defined, although this author describes only his own use of the Index Technology Corporation products.

Extending an existing CASE environment. The strategy defined here revolves around creating extensions to an existing environment. To successfully integrate a new methodology, the tool features that must be addressed are menus, entity support, relationship support, screens and data capturing, and methodology rules support. Each of these features is addressed in this paper.

The logical structure of an existing environment is depicted in Figure 1. In this environment, users

Figure 3 Process for implementing box definition graphs


LEARN THE METHODOLOGY
IDENTIFY THE COMPONENTS
CHOOSE THE CASE ENVIRONMENT
LEARN THE CASE ENVIRONMENT
IMPLEMENT THE METHODOLOGY

access menus to create deliverables (e.g., graphs) of the various methodologies supported. The user also follows the built-in definitions that determine how the methodology is to be used, how its components (or entities) are stored in the environments dictionary, and how the methodology constructs are related.

Figure 2 depicts an extended CASE environment. New methodology support has been added and the environments dictionary has been extended to allow for creating and storing instances of entities defined for the new methodology. Additionally, new relationships have been defined which allow a user to create and store links between previously existing methodologies and the newly defined methodology.

The paper next addresses the process required to effectively implement the extended environment strategy.

BSM tool support procedures

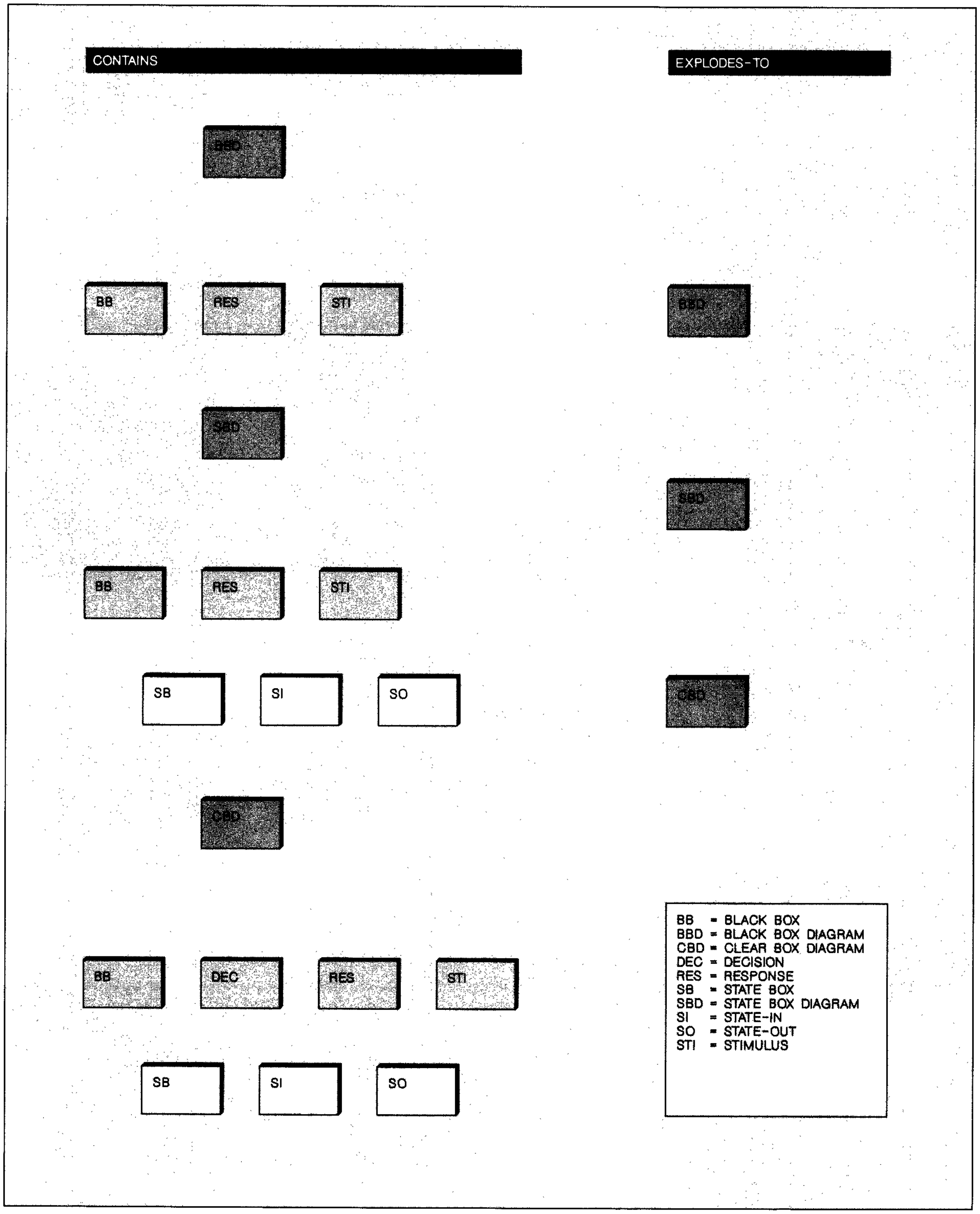
Figure 3 illustrates a process consisting of five basic procedures that have been followed to successfully implement BDG in an existing CASE environment. These are now discussed.

Learn the methodology. There are several major areas that should be of principal consideration in supporting a new methodology within an existing environment.

The goal of learning the methodology to be implemented requires one to be able to identify the entities and relationships which make up the methodology. By identifying the entities and relationships, the tool builder will have, in effect, developed a scheme to be used in actually implementing the methodology. The entity-relationship diagram for BSM shown in Figure 4 provides an example of how this helps in learning and understanding BDG.

Most of the steps and procedures described are independent of which environment has been chosen for implementation. Past experience has shown that learning the methodology may take as much as 80 percent of the time required to implement tool support for the methodology. The remaining time is

Figure 4 Entity-relationship diagram representing BSM



involved with learning about a particular environment and its customization functions.

An important step in developing tool support is to completely understand how that methodology is used. This involves reading available literature on a methodology such as box structures and actually using it to address a sample problem.[1,2] The tool builder can then determine the effect of automating the methodology using the tool support. Most of the reference materials written on methodologies include examples that show the reader how to apply the methodology. It is important for the tool builder to follow these examples, since they will provide insight into ways the methodology has been successfully applied and where tool support is most required.

While using the new methodology, tool builders will develop ideas on how they think it should be implemented in conjunction with understanding how the chosen CASE environment supports its standard set of methodologies. In this case, BSM diagramming techniques need to be studied to determine how they might be used in conjunction with data flow diagrams, entity-relationship diagrams, and data modeling.

Identify the components. Once it is learned how BSM is used to describe a system, the tool builder must now develop an underlying understanding of what the components of BSM are and how they relate to one another. The most efficient method of doing this is to create an entity-relationship diagram for BSM (see Figure 4). The following describes the preliminary work to define the entity-relationship diagram.

There are two basic types of entities in a software engineering methodology. The first type, a *simple entity*, cannot be decomposed and is completely atomic. The second type, *complex entities*, are those which can be thought of as containing or being composed of one or more simple entities.

A study of BSM produced this list of simple entities:

- Black box
- State box
- Decision
- Stimulus
- Response
- State-in
- State-out

Further study determined that BSM consists of the following complex entities:

Table 1 Contains relationships

Black box diagram	Contains	Black box Stimulus Response
State box diagram	Contains	Black box Stimulus Response State box State-in State-out
Clear box diagram	Contains	Black box Stimulus Response State box State-in State-out Decision

Table 2 Explodes-to relationships

Black box diagram	Explodes-to	State box diagram
State box diagram	Explodes-to	Clear box diagram
Clear box diagram	Explodes-to	Black box diagram

- Black box diagram
- State box diagram
- Clear box diagram

The next step is to determine the relationships between these entities allowed by the definition of BSM.

Two important types of relationships are defined: The *contains* relationship indicates that a complex entity can contain (or be composed of) another entity. The *explodes-to* relationship indicates that an entity (of either type) may be described in further detail by another entity (usually a complex entity). Given these relationships, we determined that BSM is made up of the contains relationships shown in Table 1. BSM also allows the explodes-to relationships shown in Table 2.

These core relationships define BSM and are the minimum relationships which must be supported to adequately aid in the creation of BDG diagrams. Other relationships may be created which relate entities of BSM to entities of other methodologies.

Choose a target CASE environment. The topic of choosing a CASE tool environment has been covered in many papers that also describe how to match an implementation with the user's methodology needs. Since this paper deals with CASE environment customization, the focus here is on requirements that must be fulfilled to be able to quickly add new methodologies to an extendable environment. The following paragraphs discuss a required list of functions that should be considered in addition to the basic common functions. Each required function includes a brief description of the aspects of the environment that it affects.

This list of functions is not meant to be conclusive, but should give the reader an idea of the critical aspects of an environment that should be capable of being customized. The list can then be used to compare different CASE environments.

Standard set of methodologies. Environments should contain a standard set of methodologies that include support for data flow, control flow, and modeling. This allows tool builders to spend their time enhancing these standard methodologies or creating new methodologies not in the standard set. The environment should allow the user to create relationships between existing methodology support and new methodology support created as a result of extending the environment. These relationships can then be followed to trace the transformation of information described.

Creation of new entities and shapes. To create tool support for new methodologies, the environment must allow tool builders to define new entities to the dictionary that the environment maintains. The new entities that are created should have the same support that any existing entities (included in the standard set of methodologies) might have.

The environment must support the creation of user-defined shapes that will represent the physical attributes of the entities that they have defined. Primitive shapes should be provided to the tool builder from which to make simple modifications and create customized shapes and symbols.

Screen and menu customization. Screen customization that captures information is a necessary function. Tool builders should be able to use functions of the environment to alter the content and format of any screen used by the tool. Additionally, the environment should allow the creation of any new screens required by the introduction of new methodologies.

Menu customization must be allowed. The environment should also allow the user to create new menus, menu hierarchies, and chains of menus and screens. Tool builders should be given functions that allow them to view the existing menu hierarchies and change them to fit their requirements for new methodologies and extensions to existing methodologies.

Existing users must be given a simple migration path.

Analysis extendability. A critical function of any environment is the ability to analyze the dictionary elements created as a result of using the environment. This type of analysis involves using the methodology rules to check for completeness, accuracy, and consistency. Analysis can be categorized into two types, static analysis and dynamic analysis.

Analysis consisting of rules that check the usage of the methodology, that is performed after the methodology has been used, and occurs after the tools dictionary has been populated with design information is called *static analysis.* These rules are invoked by users whenever they decide that they have a sufficient level of information to begin verification. Invocation of these rules takes place outside of the diagram editing environment, usually by the creation of a report. A user-definable query capability should be provided by the environment to allow users to generate reports that meet their unique needs.

Static analysis and *dynamic analysis* can have the same rules. The distinction between the two types of analysis is in the way that the rules are invoked. With dynamic analysis, the environment parameters determine when to invoke the rules that evaluate the correctness of what has been entered. This would allow the identification of an error as it was made, such as in the diagram editing environment. For an environment to support the customization of dynamic analysis rules, functions must be provided to the tool builders, thereby allowing them to create and change rules for new or existing diagrams and components of diagrams.

Migration of customized product. Existing users of the CASE environment must be given a simple migration path to utilize the new features and functions of any new customized versions. The environment should not require the user or tool builder to write additional functions to perform any migrations.

As tool customizers become increasingly popular, more and more unique versions of customized environments will be developed. Since some of these environments will have features and support that others do not have, the requirement to merge the various versions becomes critical. Additionally, integration with existing tools is a necessary function. The CASE environment should employ an architecture that fosters enabling the environment to allow users to invoke their own functions and tools. A prime example of this would be the support for allowing a PC-to-host communication product to remain active while the CASE environment is running. This provides the user with the capability of switching easily to the host session and performing functions on the host.

Printing support. Support should be provided for creating embedded files which can become part of large design documents. New graphs should have the same workstation printing support as graphs from the standard set of methodologies found in the environment. Diagrams and other dictionary elements should be capable of being exported from the environment in a format which can be converted and printed on a host-connected page printer. The dictionary should allow a fast easy way of exporting dictionary elements to support tool builders in writing their own functions to export the elements, transfer them to the host, and transform them to the required printer format.

Programming interface. No environment can anticipate all the possible ways in which users will want to access and use their design information. To allow for customized access to the design information in the dictionary, a programming interface is required. This programming interface would define functions and methods for retrieving stored information, such as instances of new and existing entities.

Learn the target CASE environment. Once an environment has been chosen, the tool builder must become familiar with the end-user functions provided. This involves learning the environments dictionary capabilities, standard methodology support, user interface functions, and menu structure. The following describes these areas in further detail.

A most important aspect of an environment is the support provided to capture and maintain information in a dictionary. Characteristics of a dictionary include *entity definition* where the components of a methodology must be stored as distinct entities that have attributes and relationships also maintained in the dictionary, and *entity reporting* where the dictionary should allow the retrieval of the attributes and relationship information.

The tool builder must understand the level of support provided for the creation and maintenance of entities. To do this, the tool builder must use the functions of the environment that support entity definition.

Most environments have a standard set of methodologies that cover some subset of the software development life cycle. For tool builders to implement a new methodology in the environment, they must first understand the existing methodologies and how they are meant to interact and relate to each other. This includes understanding which existing methodologies should connect and tie into the new methodology. Additionally, the tool builder should note any similarities between existing and new methodology constructs, so that aspects of these constructs can be reused in the new methodology.

The user interface defines the look-and-feel of the user's interactions with the environment. Tool builders should become familiar with the following user interface related functions:

- Zooming allows the user to physically view a graph at different levels of detail.
- Scaling allows the user to change the size of the shapes that represent a methodology construct.
- Line drawing represents connections, inputs, or outputs and how those lines are redrawn whenever the methodology construct to which they are connected is moved or deleted.
- Text labeling allows the user to enter and view text associated with methodology constructs (for example, the user-defined name of a shape).
- Scoping allows the user to select methodology constructs to make them the focus of an operation, such as moving or deleting.
- Delete verification allows the user to be prompted for verification before a delete is actually performed.

In general, any new methodologies will have a user interface similar to the standard one provided by the environment. Trade-offs and shortcomings in the standard methodologies will be present in any new methodology implemented with the chosen environment.

The menu structure of the environment defines the menus from which the environments functions are selected. The tool builder needs to understand all of the menus which could be affected as a result of the addition of a new methodology. Additionally, the tool builder may wish to rearrange the existing menu structure. This would allow tool builders to replace or delete existing menu entries for functions of the environment that their target users do not use.

Customization is disjoint from end-user functions.

Implement the methodology. At this point in the process, the tool builder should have a documented description (by following the previous procedures) of the new methodology. The tool builder will also have an understanding of the capabilities of the environment selected. With this information, the tool builder can then begin to implement support for BSM using the customizing functions of the CASE environment.

So far, the procedures in this process have not been specific to the selection of a particular environment. During this step, however, the tool builder will begin to use functions which are specific to each individual environment implementation. The tool builder will begin by following the documentation provided by the environment and must determine exactly how to define the following types of information: menus and menu flow, entity definition, relationship definition, physical definition of entities and relationships (shapes), data capturing screens and their flow, and methodology rule enforcement.

Existing CASE environments that provide customization have that function disjoint from the actual end-user functions. This prevents the user from creating unvalidated versions of tool support for the methodology. Once an implementation is created and tested, the last step is to validate the new tool support. This includes using the newly created tool to enter and analyze a sample problem. While this can be done by the tool builder, it is best to have the validation step performed by another group, such as one responsible for educating new users of the methodology. It is also beneficial to include people from the user organization who may have application-specific insight into how the methodology will be used. As a result of validation, the implementation may require changes to the methodology to make it more amenable to computer-aided tool support.

BSM tool results

By following the strategy and procedures defined in the previous sections, computer-aided tool support for the box definition graphics component of the box structure methodology was created. This section describes this tool support and explores its current and future use.

Using an existing tool environment. A commercially available vendor tool environment (Excelerator® and Customizer) was utilized to create the BSM tool support described in this paper. The customizer component of the tool environment (Customizer) enabled the creation of new functions required for BSM, and the run-time product (Excelerator) was then used to test the newly added functions.

The principal reason for choosing an existing CASE environment for this exercise was the need to quickly create prototype BSM tool support and evaluate its impact on increasing the effectiveness and acceptance of the methodology. Since the BSM user community required immediate tool support, creating a specific stand-alone BSM environment was not an option. By customizing an existing CASE tool environment, a prototype could be developed quickly and different versions of the BSM support could be compared to determine which version best supports the methodology. Additionally, by using an available environment, several distinct advantages are obtained.

One advantage with this strategy is the ability to integrate BSM with other methodologies. For example, by integrating BSM support in Excelerator, users could use the entity-relationship support to create entity-relationship diagrams for their system. Following this exercise they could then create box definition graphs that could be linked to the entity-relationship diagrams. This would provide the users with two powerful views of their system, with the connections between these views maintained by the tool environment. With this strategy, the BSM data are stored in the same dictionary as the data created from using other methodologies. This allows the user to reuse this information in the creation of graphs. Additionally, the information defined in the graphs can be reused in other methodologies.

Another advantage is the seamless environment that is presented to the user. When the CASE tool user sees the BDG support, it looks and feels like the same support provided for the other methodologies in the tool set. This common user interface provides a large degree of productivity since the user of BDG does not have to learn a new interface to be able to enter BDG data.

Not all of the advantages of this strategy are tied directly to BSM. The fact that Excelerator is an established tool with a large user community is also an advantage. This wide user acceptance means that there is a large set of potential users who are already using the methodologies supported by Excelerator with help and installation support in place. Additionally, classes are available that educate the new Excelerator user about the user interface and base dictionary support.

Excelerator with BSM. The following paragraphs describe the various components of Customizer and how they were used to create a customized version of Excelerator that supports the entry and analysis of box definition graphs.

Defining the BSM shapes. Each of the simple entities of BSM has a physical shape associated with it. These shapes were defined in the document which introduced BSM.[7] One of the first steps in creating a customized version of Excelerator is to define these shapes using the Customizer shape editor that supports the creation of customized shapes used to represent the entities of a graphical methodology. With the exception of a diamond shape for the decision entity, the shapes corresponding to the other BSM entities are simple boxes.

Defining BSM entities and relationships. Most of the work defining entities and relationships utilized functions of the Customizer system dictionary. Excelerator menus were extended to include options for box structures constructs. This includes allowing the user to choose to work with black box, state box, and clear box diagrams from the main menu. Additionally, new menus were created for each of these types of diagrams containing the constructs available for use in that diagram. Other Excelerator menus were also extended to include the BSM entities and relationships. An example of this would be the menus that allow the exporting of Excelerator data.

The Excelerator dictionary (XLDictionary) was extended to include the definition of the entities described during the "learn the methodology" phase of the procedures. These entities were defined and, where appropriate, matched to shapes created with the shape editor. The black box, state box, and decision entities all had corresponding shapes. The stimulus, response, state-in, and state-out entities were defined with the Customizer as connections, with stimulus and response being represented by straight lines and state-in and state-out represented by dotted lines. XLDictionary was also extended to include the definition of the relationships.

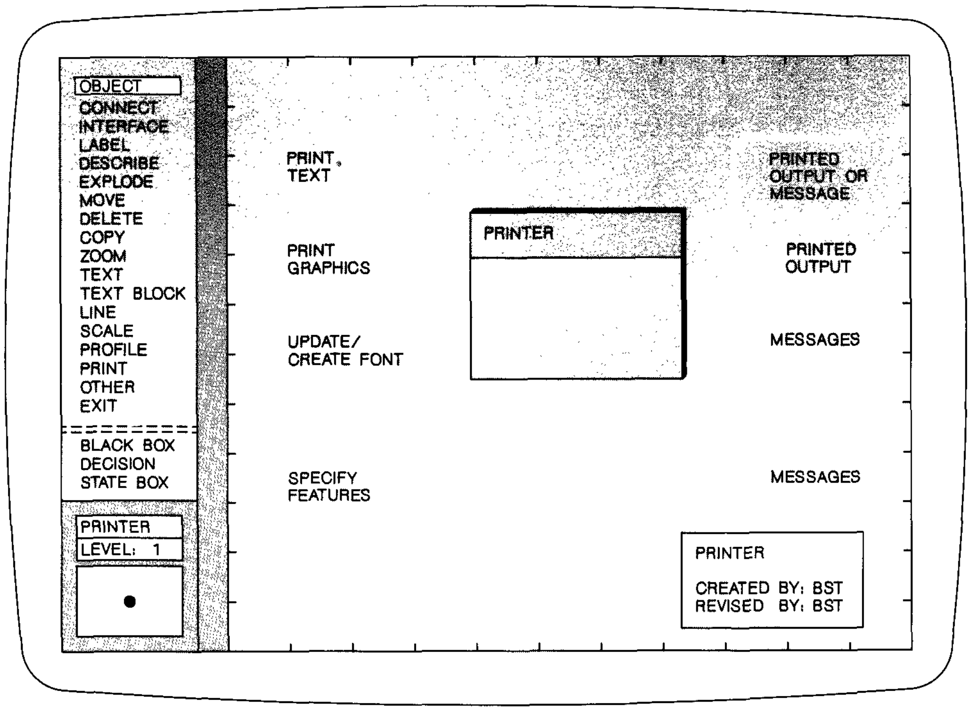
The contains relationships were created implicitly by allowing the entities to be available on certain menus. The explodes-to relationships were created using a standard Customizer explodes-to screen. This allowed explodes-to relationships to be created in two directions, from BSM to other methodologies and from other methodologies to BSM.

Validation with methodology educators and potential users

After the extended version of Excelerator was created, a process was performed to determine the validity of using the tool to create box definition graphs. To maximize the effectiveness of this process, a representative from the area responsible for educating developers in the use of BSM and representatives of potential users of the new BSM support were included to ensure that the new tool environment not only supported the methodology from a theoretical view but also from a practical view.

At the core of the validation process was a real life problem for which box definition graphics had already been created without the use of a CASE tool environment.[2] The problem was represented in the extended environment using the newly created tool (Figure 5) and the computer-assisted BSM support was then analyzed and changed to more closely

Figure 5 Excelerator with box definition graphics



match the methodology and to improve on its ease of use.

Once the customized Excelerator prototype had been validated, education, tool-usage procedures, and on-line help text was then developed. These procedures were developed in a cooperative effort between the area responsible for methodology education and a user representing the potential user community.

Finally, the customized Excelerator environment and the education material were successfully used on a pilot project for a computer-integrated manufacturing system being created by the IBM Application Solutions Division.

What was learned. Several important results were derived from the creation of tool support for BSM. Most important was the validation of the strategy of using a tool customizer to quickly create effective tool support for BSM. The steps involved in implementing BSM with a tool customizer took only several days, once the strategy and procedures were defined and followed. After the methodology had been mastered, the time to create different versions of tool support was minimal, providing the opportunity to choose the best implementation.

Since BSM was a new methodology that had not been used extensively, there was no definitive example or precedent detailing its use. This became apparent in the development of the entity-relationship diagram for BSM. While performing this part of the process, it was evident that the available literature was lacking in helpful examples. To solve this shortcoming, an area responsible for technical education assisted in the determination of those decisions in tool support resulting in the best tool implementation.

When questions arose regarding tool support for BSM, they usually originated from an incomplete or inaccurate understanding of the methodology. From this we learned that the most important step of the procedure is learning the methodology. During this step it was very important to get the experts in the methodology involved to ensure that any ambiguities in the definition of the methodology were clarified and that a solution was agreed upon.

Another outcome of this study was the familiarity gained with tool customizers. It was learned that these customizers have extensive capabilities, allowing almost every aspect of the CASE tool environment to be tailored.

Having successfully implemented tool support for BSM, other methodologies are now being considered for implementation. These methodologies could be included in the same CASE tool environment as was BSM support, allowing potential users more options in choosing methodologies that match their problems and work well together. An example of this is the current study to develop support for the integrated computer-aided manufacturing definition model (IDEF0)[8] methodology. This methodology is used to model enterprise data and functions and can be used in conjunction with BSM.

Conclusion

Development of computer-aided tool support for new software engineering methodologies can be achieved quickly and efficiently once a strategy and set of procedures is developed and followed. Most of the tool builder's time is invested in becoming familiar with the methodology and a CASE environment to support it. Implementing a new methodology is actually the most straightforward task and takes the least amount of effort.

The available tool customizers evaluated are both powerful and easy to use. By choosing to create tool support for BSM in existing environments, several implementations of BSM can be created, reviewed, and validated in a short period of time by following the procedures described in this paper. As tool support for BSM matures, the methodology may increase in popularity and acceptance, allowing it to become more widely used in the development community.

The tool support that was described in this paper (Excelerator with BSM) is currently being used internally in IBM to develop systems using the box structure methodology.

Acknowledgments

The author would like to thank Jack Odom, Len Marchese, Bruce Sicherman, and Janet Slyman for their roles in the work described in this paper. A special thanks to Jay Friedman for his tireless and careful editing of the original version of this paper.

Cited references

1. H. D. Mills, R. C. Linger, and A. R. Hevner, "Box Structured Information Systems," *IBM Systems Journal* **26,** No. 4, 395–413 (1987).
2. J. E. Odom, "Using Box Structures for Definition of Requirements Specifications," *IBM Systems Journal* **29,** No. 1, 59–78 (1990, this issue).
3. P. Judge. "Support Stage for Analysts (CASE Tools Survey)," *System International* (Surrey, England) **17,** No. 3, 37–42 (March 1989).
4. R. J. Norman and J. F. Nunamaker, Jr., "Integrated Development Environments: Technological and Behavioral Productivity Perceptions," *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, Vol. II: Software Track,* IEEE Computer Society Press (1989), pp. 996–1003.
5. P. N. Robillard, "On the Evolution of Graphical Notations for Program Design," *ACM SIGSOFT Software Engineering Notes* **14,** No. 1, 84–88 (January 1989).
6. C. R. Necco, N. W. Tsai, and K. W. Hogelson, "Current Usage of CASE Software," *Journal of Systems Management* **4,** No. 5, 6–11 (May 1989).
7. H. D. Mills, R. C. Linger, and A. R. Hevner, *Principles of Information Systems Analysis and Design,* Academic Press, Inc., New York (1986).
8. "Integrated Computer-Aided Manufacturing Final Report: IDEF0 Functional Modeling Manual," U.S. Government Contract No. F33612-73-C-5158, SOFTECH Inc., Waltham, MA (January 1981).

Bradley S. Tagg *IBM Enterprise Systems Division, P.O. Box 700, Suffern, New York 10901.* Mr. Tagg is a staff programmer who designs and implements IBM computer-integrated manufacturing (CIM) systems. He graduated from Ohio State University in 1982 with a B.S. in computer and information science, and joined IBM in 1984 as a programmer. He has worked on a variety of development assignments, including IBM internal computer-aided software engineering (CASE) and CIM systems.

Reprint Order No. G321-5387.