Development of a Comprehensive Software Engineering Environment

Thomas C. Hartrum
Department of Electrical and Computer Engineering
School of Engineering
Air Force Institute of Technology
Wright-Patterson AFB, Dayton, Ohio, 45433

Gary B. Lamont
Department of Electrical and Computer Engineering
School of Engineering
Air Force Institute of Technology
Wright-Patterson AFB, Dayton, Ohio, 45433

Abstract

The generation of a set of tools for the software lifecycle is a recurring theme in the software engineering literature. The development of such tools and their integration into a software development environment is a difficult task at best because of the magnitude (number of variables) and the complexity (combinatorics) of the software lifecycle process. An initial development of a global approach was initiated at AFIT in 1982 as the Software Development Workbench (SDW). Also other restricted environments have evolved emphasizing Ada and distributed processing. Continuing efforts focus on tool development, tool integration, human interfacing (graphics; SADT, DFD, structure charts, ...), data dictionaries, and testing algorithms. Current efforts are emphasizing natural language interfaces, expert system software development associates and distributed environments with Ada as the target language. The current implementation of the SDW is on a VAX-11/780 under VMS. Also, a simplified version of the SDW has been hosted on personal computers. Other software development tools are hosted under UNIX and are being networked through engineering work stations. This paper discusses the various aspects of AFIT's development of software engineering environments.

Introduction

A software development environment is an integrated set of automated and interactive software development tools that aid the software engineer in the development of quality software products. The specific software products which are associated with the software life cycle include requirements definitions; design specifications; source and executable program codes; test plans, procedures, and results; as well as other associated documentation such as guides and manuals of operation and maintenance of the software. By definition, software only exists in its documentation! Thus, extensive records must be generated, maintained, and managed to properly fulfill the software engineering objectives. A well planned and implemented software development environment can effectively assist in the generation of reliable and maintainable computer software.

The typical software development environment includes both hardware and software tools to aid the software engineer in the production of programs. Software development environments may consist of a minimal set of tools, such as an editor, a compiler, and a link/loader, that support only the actual coding of software. However, the most effective environments are those with an extensive set of powerful interactive and integrated tools that support state-of-the-art methodologies for dealing with software from its very conception through its eventual termination. A specific software development environment consists of a process methodology along with given hardware and system software, manual procedures and support personnel. The process methodology usually involves a specific set of operations (steps) along with conceptual tools to support these steps within the software lifecycle phases mentioned previously.

The concept of an integrated software development environment can be realized in two distinct levels. The first level deals with the access and usage mechanisms for the interactive tools, while the second level concerns the preservation of software development data and the relationships between the products of the different software life cycle stages. The first level requires that all of the component tools be resident under one operating system and be accessible through a common user interface. The second level dictates the need to store development data (requirements specifications, designs, code, test plans and procedures, manuals, etc.) in an integrated data base that preserves the relationships between the products of the different life cycle stages. This integration of tools and techniques at both levels is a major objective of any software engineering environment development effort.

The major objectives are to provide a production software development environment for students and faculty and to generate a software engineering research testbed. Initially the SDW provided the overall architecture for a "complete" capability. Recently, efforts are focusing on a distributed version of the SDW concept called System 690 in support of the software engineering laboratory course, EENG 690.

Development Lifecycle Model

The definition of the software lifecycle as supported and used by the various environments consists of the standard six phases; requirements definition, preliminary design, detailed design, implementation (coding), integration and operation and maintenance. This general methodology is reflected in DOD Standard 2167 [1]. Documentation must be provided within each phase to support reviews (static) and testing (dynamic) of results associated with each activity. This capability can be provided through the use of a data dictionary and associated data base management system. Software system corrections and enhancements should flow through all previous phases for "proper" documentation.

Note that validation and testing is not a distinct stage in this lifecycle, but rather an activity that is performed along the entire lifecycle. This activity involves the testing of the products of each stage for internal consistency and completeness with the products of the previous stages. Furthermore, the products of each stage are validated against the user's perception of the requirements.

The Software Development Workbench(SDW) and the distributed environment, System 690, are developed using this software life cycle definition with the primary objective of providing integrated and automated support. Discussion of each environment follows the stages of its initial lifecycle. The objectives and accomplishments of each stage of the lifecycle development are presented. The requirements definition and preliminary design stages deal with a system as it should exist in its ultimate form, whereas the detailed design, implementation, integration, and operation stages emphasize a prototype environment with a menu driven interface and initial tool set.

SDW Requirements and Design

The first stages of the SDW development effort [2] emphasized the requirements definition and preliminary design of the ultimate SDW implementation. Due to the extensive scope of this task, the target was set at a fairly high level specification with the individual subsystems specified in greater detail with the use of recursive applications of the software life cycle.

The results of this task are a set of five primary objectives of the SDW, thirteen specific concerns for its development [3], a functional model and associated evaluation criteria, a hardware/software configuration model, and a structure model that identifies all generic component tool types.

Of the five objectives of the SDW, the reduction of software errors is the first [4]. This is to be achieved by supporting and enforcing the use of accepted software engineering principles, as well as by using the computer to augment different testing procedures.

The SDW must also be responsive to change. Realizing that software is a dynamic entity, the SDW must be able to support changing requirements for its operation.

The rapid assessment of design alternatives is also quite important. The use of simulation models and prototyping is selected more and more to assess design operations as well as to aid in determining the end user's true needs.

The SDW must also be capable of providing interactive and automated documentation support. This support must emphasize the recording, and maintenance of all software development associated data.

Finally, the SDW must provide mechanisms to assist the software manager in planning and tracking software development efforts.

The thirteen specific concerns also required to be addressed by the SDW development effort are: integration, traceability, user-friendliness, testability, pre-fabricated programming, support for the entire software lifecycle, flexibility, consistency and completeness, explicitness and understandability, documentation support, updateability, language independence, and early prototyping. The first five of these concerns are of special significance to the SDW effort. Integration is to be realized in terms of both accessing component tools and storing of the development data. Traceability must also be preserved between the products of the different stages of the development effort. User-friendliness is also a very significant concern. The SDW must utilize the latest concepts of ergonomics in the design of its human interface. This interface should be easily understandable with a simple logical structure, well laid out display, and a simple command input mechanism. Prefabricated programming, or the incorporation of existing software can improve development productivity. Flexibility is required at both the environment and tool level to allow users to operate in a mode comfortable to their knowledge and experience levels. That is, the operation of the SDW must allow the user to tailor the type of prompting, feedback, and structure [5].

A functional model of the software development process using SADT (Structured Analysis and Design Technique) diagrams [6] was developed in order to define the SDW process and to select those aspects of the process that could be automated. Furthermore, a set of evaluation criteria is established with which to judge the effectiveness of the environment in satisfying its requirements. However, for reasons of brevity, these topics are not discussed further.

The configuration model of the SDW shown in Figure 1 illustrates the basic hardware/software configuration for the environment. The SDW Executive is the primary interface and controller of the component tools. The SDW tool set is broken down into three tool categories; cognitive tools, that extend the powers of understanding for the software developer; notational tools, that assist in the production and maintenance of associated documentation; and augmentive tools, that use the powers of the computer to perform much of the tedious testing and updating activities involved with software development. The project data bases are the integrated data storage areas with one allocated to each development effort. Finally, the Pre-Fab Software Description and Product Data Bases hold the functional descriptions and program codes of existing software modules. This structure provides for easy retrieving and incorporation of modules into development designs and implementations.

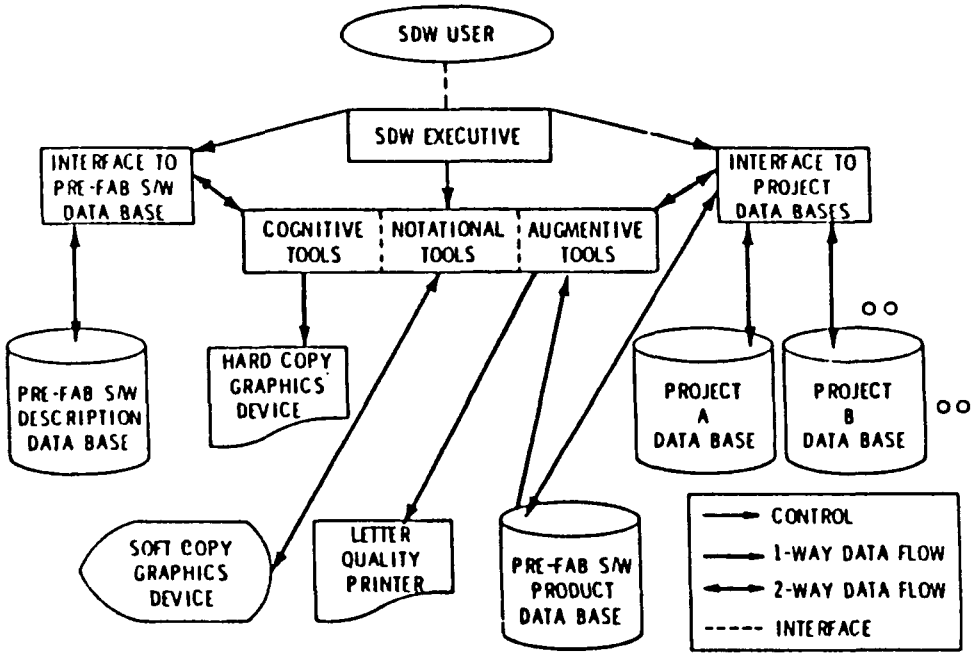

Figure 1: SDW Configuration Model.

The structure model of Figure 2 illustrates the generic tool types that are to be incorporated into the SDW. Those tool types annotated with a single asterisk are included in the initial implementation of the SDW, while those with two asterisks are scheduled for the second level of implementation. Those with three asterisks are to be included as they are developed or become available. Thus, the frameworks for the initial and eventual realizations of the SDW are established. With this background, a detailed design and implementation can be realized that includes the selection of existing tools as components and complete development of other components.

Current Implementation of the SDW

The detailed design, implementation, and integration stages of the SDW development effort focus on the accomplishment of an initial version of the environment. This initial version is composed of software development tools that support the pre-implementation activities of software development as well as provide the common capabilities found in most implementation oriented development environments such as editing, linking, and debugging.

The discussion of this initial version of the SDW is limited to two topics: the selection of an initial tool set and the complete development of the SDW Executive (SDWE) component that provides the common access and control mechanisms required to satisfy the first level of the integration criteria.

The tools selected for inclusion into the initial implementation of the SDW are taken, for the most part, from one of two sources. Those tools that specifically support the requirements specification and design activities were given by the Integrated Computer-Aided Manufacturing Division of the Air Force's Material Laboratory, Wright-Patterson Air Force Base. The tools used to provide the rest of the development support are the standard vendor supplied tools normally found on the target computer (the Digital Equipment corporation's VAX-11/780 under the VMS operating system).

Four distinct tools are selected to support the first two phases of software development. They are the AUTOIDEF [7], that supports the Integrated Computer-Aided Manufacturing (ICAM) Definition Techniques (IDEF) [8], the SYSFLOW graphics editor [9], and the Extended Requirements Engineering and Validation System (EREVS) [10].

The AUTOIDEF tools support and aid in the production and maintenance of three types of IDEF models. IDEF-0 models are used to provide a functional modelling capability which describes the flow of data through functional processes. IDEF-1 models provide an informational modelling technique that describes both the components of a data entity and the relationships between data entities. Finally, IDEF-2 models are used for dynamic modelling to simulate transaction flows through network-like systems. The AUTOIDEF tool greatly simplifies the production and maintenance of all three types of models because of its flexible graphics drawing and modification capabilities.

The SYSFLOW graphics editor is an easy to use and flexible tool. The tools provide a basic set of graphical constructs and character fonts, together with the capability for the user to define his own constructs, to provide a very flexible capability to produce and maintain a great variety of graphical/textual documentation. This system can be employed in generating data flow diagrams defining detailed requirements or it can be used to define structure diagrams as generated by transform analysis or transaction analysis [11] of the requirements in a data flow or SADT format.

The ICAM Decision Support System (IDSS) provides for the graphical and textual input of IDEF-2 dynamics models. The results of executing these simulation runs are analytical reports on the simulated system's performance. The provisions for graphical input of models and automatic translation into an executable format make the tool a truly state-of-the-art facility.

The Extended Requirements Engineering and Validation System (EREVS) was originally developed by TRW, Inc. for the Army's Ballistic Missile Advanced Technology Center. EREVS provides sophisticated facilities for specifying system requirements for concurrent and real time systems, checking those requirements for consistency and completeness, illustrating the requirements with a graphical technique

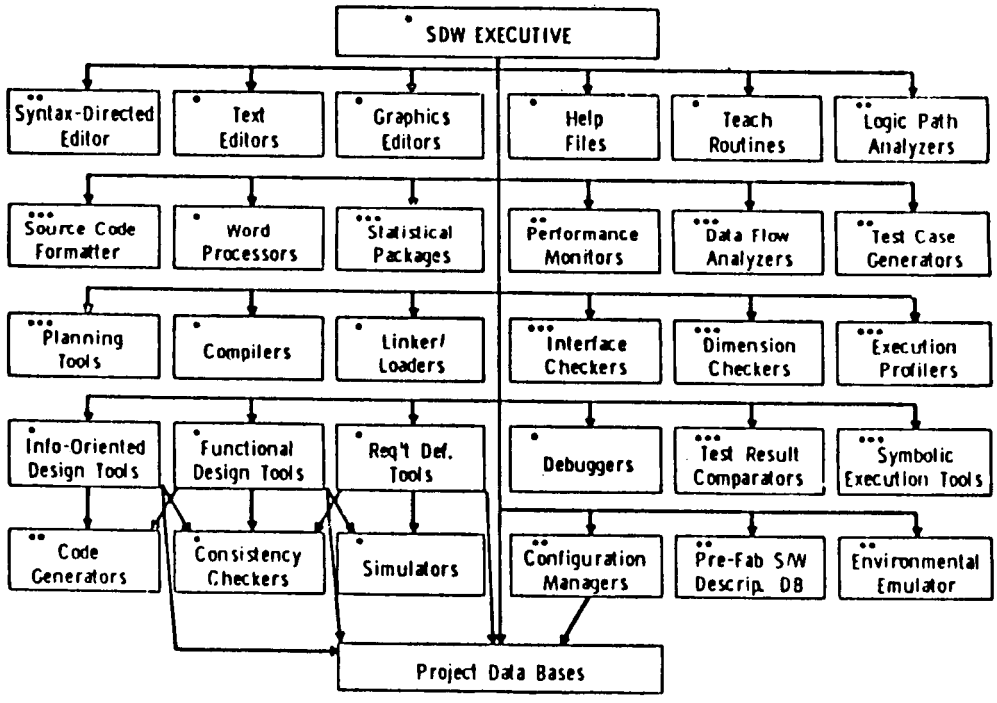

Figure 2: SDW Structure Model.

called R-nets, and then simulating the timing feasibility of the stated requirements. Although specifically designed for concurrent and real time software systems, EREVS is an effective aid in developing well stated and feasible requirements for all types of software systems.

In addition to these tools, the SDW uses the standard VMS available tools to perform the required compiler, linker, editor, debugger, comparator, and text processing functions. Moreover, DEC's Program Development Tools consisting of programming tools and project management tools can easily be integrated into the SDW. The programming tools include the language-sensitive editor (syntax-directed), the source code analyzer, the symbolic debugger, and the performance and coverage analyzer (PCA). The project management tools include the code management system(CMS), the module management system(MMS), the test manager and the common data dictionary(CDD). Other DEC software products supporting software development include their data-base management systems (DBMS, Rdb/VMS, Datatrive), forms management system(FMS), and the application control and management system(ACMS).

With this set of components established, the SDW Executive is developed with common interfaces to specific tools. The SDWE is a menu and command driven interface to all of the SDW component tools but also provides access to all of the facilities provided by the VMS operating system.

In order to structure the accessing of the SDW component tools, these tools are assembled into groups by related functions. The top level menu of the SDW allows for the selection of any of these functional groups. Once a functional group is specified, all of the member tools of that group can be accessed through a new menu.

At any level in the hierarchy formed by these menus, any of the standard VAX DCL commands may be executed. Furthermore, multiple levels of the menu hierarchy may be traversed at any time by simply entering the appropriate command string on a single command line.

On-line help facilities are provided for all levels of the hierarchy. These help facilities provide either general help on the environment, specific help information on any of the currently accessible commands, and access to the VMS standard help facility. Additionally, utility functions are provided to enable/disable the automatic displaying of current menus, to change the type of terminal in use, and to alter the manner in which the development data is stored.

Specific SDW component tools may either by accessed through the menu structure and command or by using special commands to execute the tool directly, thus increasing the flexibility of the environment.

Development data storage is provided by this implementation by establishing isolated data storage areas for each supported development effort. However, these data storage areas do not at present provide for the full integration of the data that is defined by the preservation of the relationships between the different development products.

The SDW Executive was also designed to be easily modifiable. Thus, new tools may be easily incorporated into the environment. Furthermore, a full set of documentation is provided on the SDWE. This documentation includes a user manual, an installation guide, and a maintenance guide that is to be used to modify and tune the environment for specific applications. The SDW is currently installed on the AFIT research VAX-11/780. Users have found the environment to be very easy to learn and use.

Expert Systems and Software Engineering

The integration of artificial intelligence concepts into software engineering environments currently focuses on expert systems. Specific expert subsystems must be developed for each phase of the software lifecycle to assist in design development and selection, structural formulation, algorithm determination, structured programming implementation (object-oriented, abstract data types, control structure), module and system testing and maintenance. An initial effort towards defining an associated environment resulted in a modification of the SDW executive using OPS5 for expert system inclusion [12]. Also, this initial design focused on the analysis and diagnosis of modules in terms of coupling and cohesion standards.

Another aspect of AI integration into software engineering environments was the development of a natural language interface [13]. This activity generated a natural language interface called "COIN" which uses Lisp and the Flavors package. This initial effort emphasized the interface to the data dictionary (DD) package mentioned previously since the perspective user would have a considerable dialog with the DD in defining detailed entries and preforming queries.

Additional efforts involve the use and analysis of transformational systems that can encompass knowledge-based capabilities for software production. Example efforts include the Knowledge-based Software Assistant (KBSA) and REFINE, a wide spectrum language for the development phases of the software lifecycle. Incorporation of wide-spectrum languages into a software environment may be feasible and economical which could be part of environment enhancements.

SDW Enhancements

The first enhancement is to extend and refine the SDW tool set to provide a full array of capabilities to support the entire software life cycle. This tool set must also be refined so that only those tools that are truly effective and useful remain part of the environment. Also, a user should be able to specify that only a certain sequence of tools be used in a given project and the SDW would provide only that environment, such as for Ada real-time applications.

The Pre-Fab Software Description and Product Data Bases must be completely developed and populated to support the prefabricated programming concept. After the establishment of a fairly static tool set, a schema for the Project Data Bases can be developed. These data bases will hold all of the development data for the products of each software development effort as well as the relationships between the different products of each effort. An initial project in this regard generated a prototype data dictionary [14] for the SDW using the DBMS Ingress. This effort was further enhanced with the System 690 project under the UNIX operating system.

The scope of the support provided by the SDW is also to be expanded to aid the software development manager in planning and tracking the development effort. Responses to queries on the Project Data Bases will provide the software development manager with near real time feedback on the status of the development effort.

The current implementation of the SDW is quite flexible and an easy to use aid for the development of quality software products. This initial implementation provides extensive support for the pre-implementation stages of software development. The environment effectively increases the cognitive and notational powers of the software developer.

The ultimate implementation of the environment will support the entire software development life cycle. Much of the tedious consistency and completeness testing of software will be automated in this environment. Furthermore, provisions will be included to store and maintain all development data in a fashion that preserves traceability between the products of the different life cycle stages. Such an environment would be a significant breakthrough in the production and maintenance of quality software systems.

Distributed SE Environment

Using some of the SDW concepts, a distributed software development environment called SYSTEM 690 is being developed to support classroom and research programming projects as well as research into environment issues. SYSTEM 690 addresses the same objectives of the SDW but in a distributed environment. The computer environment used by SYSTEM 690 is both heterogeneous and quite extensive. Most of the software development is done on a network of VAXes and Sun workstations running Unix and interconnected by a TCP/IP Ethernet, and on a series of DEC VAXes and MicroVAXes interconnected by DECNET. The two networks are interconnected by a gateway. All

of these systems are also accessible via a Gandalf RS-232 switch that is connected to a variety of terminals and PCs in offices and labs, and through dial-up lines to any number of home computers. Software development is performed on these computers under a variety of operating systems using Ada, C, Pascal, Lisp, Prolog, Fortran, and assembly language.

In SYSTEM 690, specific emphasis is placed on performance monitoring and analysis to provide needed data in such areas as tool performance, tool usage, user acceptance, and the nature of the workload, both in terms of the size of data and frequency of use of the tools.

Methodology

In order to put a production software system in operation and to develop tools to support that system, the methodology selected was that mentioned in the SDW discussion, namely the use of DOD Standard 2167 and the ICAM program structure. Again, this methodology was selected with the goal of supporting the automation of the software development process, and is centered around the comprehensive data dictionary system that documents all aspects of the lifecycle as discussed previously. Each phase of the lifecycle requires its own data dictionary entities, an action entity and a data entity. Figure 3 shows an example of the information contained in these entries for the design phase. A central concept is that the data dictionary provides the complete definition of the entire development. In each of the three major phases, however, some form of graphical representation provides a more human understandable means of generating and viewing the data dictionary information. Thus the IDEF model was chosen as developed under the ICAM program. Figure 4 shows a typical analysis diagram. The underlying abstract data is stored as two types of data dictionary entries: one for each *activity* (each box on the diagrams) and one for each *data element* (each arrow on the diagrams). Information relating to the graphical layout of the structured analysis diagram is not considered part of the requirements analysis information, and is not included in the data dictionary.

For the design phase, the primary graphical representation is a structure chart. This is also documented by two types of data dictionary entries: one for each *process* (each box on the diagrams) and one for each *parameter*, as shown in Figure 3.

The design process used with SYSTEM 690 uses transform analysis and transaction analysis to evolve the requirements specification into a modular design. Detailed design is accomplished by using PDL in the algorithm section of the process data dictionary entry. Currently this is a free form psuedo-code, but in the future will be an Ada based PDL. Note that an Ada software engineering environment called ARCADE is being developed with the SDW and SYSTEM 690 efforts.

Similar to the SDW, the primary graphical representation in the implementation phase is the structure chart, representing the structural relationship between the actual code modules and showing the actual passed variables. This is also documented by two types of data dictionary entries: one for each *module* (each actual code module, subroutine, or function) and one for each passed *variable*. Of course, in this phase there is another representation of the effort, that of the code itself. The implementation process used is top-down coding, with integrated testing.

Example Data Dictionary Entry for Process

```
NAME: Process Message
PROJECT: NETOS-ISO
TYPE: PROCESS
NUMBER: 4.0.1
DESCRIPTION: Processes a NETOS message.
INPUT DATA: msgptr
INPUT FLAGS: None
OUTPUT DATA: None
OUTPUT FLAGS: error2
ALIAS: PROC-MSG
  COMMENT: Used in earlier design.
CALLING PROCESSES: Process Messages and Data
PROCESSES CALLED: Decompose Message
   Process Network 4 Messages
   Determine Channel Number
   Build Queue Buffer for Qty  = 1
   Put Buffer in Queue
   Level 4 Cleanup
ALGORITHM:
  Decompose message.
  If network message
     Process Network 4 Messages
  else
     Determine channel number
     Build queue buffer
     Put buffer in Queue
  Cleanup Level 4.
REFERENCE: PROCESS SPOOLER MESSAGE
  REFERENCE TYPE: SADT
REFERENCE: Smith's Algorithm's, pp. 23-24.
  REFERENCE TYPE: Text.
VERSION: 1.1
VERSION CHANGES: Added "Level 4 Cleanup"
DATE: 11/25/85
AUTHOR: T. C. Hartrum
```

Example Data Dictionary Entry for Parameter

```
NAME: mess-parts
PROJECT: NETOS-ISO
TYPE: PARAMETER
DESCRIPTION: Decomposed message parameters.
DATA TYPE: Composite, C structure .
MIN VALUE: None
MAX VALUE: None
RANGE OF VALUES: None
VALUES: None
PART OF: None
COMPOSITION: SRC
     DST
     SPN
     DPN
     USE
     QTY
     Buffer
ALIAS: Message Parts
  WHERE USED: Decompose Message to Validate Parts.
  COMMENT: Part of earlier design
ALIAS: messy-parts
  WHERE USED: Passed from Dump Data to Flush Buffer.
  COMMENT: Part of existing library.
REFERENCE: MSG-PARTS
  REFERENCE TYPE: SADT
VERSION: 1.2
VERSION CHANGES: Component USE added
DATE: 11/05/85
AUTHOR: T. C. Hartrum
CALLING PROCESS: Process Message
  PROCESS CALLED: Decompose Message(parts-list)
  DIRECTION: up
  I/O PARAMETER NAME: parts-list
CALLING PROCESS: Process Message
  PROCESS CALLED: Process Network 4 Messages
  DIRECTION: down
  I/O PARAMETER NAME: parts
CALLING PROCESS: Process Message
  PROCESS CALLED: Build Queue Buffer for QTY  = 1
  DIRECTION: down
  I/O PARAMETER NAME: params
```

Figure 3: Design Phase Data Dictionary Example.

A1 Execute Remote Function

Abstract: This diagram decomposes the activity of the Remote Systems into its major functions.

NOTE: This analysis assumes that a user wishing to execute a function dealing with either the Spooler System or the MSS must first establish comms with that system with a specific comms request command. An alternate approach that might be a better user interface would be for each remote function (e.g. Get File) to establish that comms link transparent to the user.

A11 Select Remote Function is visualized as being implemented by the operating system's capability of loading and executing individual programs from disk. Thus the remaining activities are seen as seperatly executable programs (although a menu-called function structure could be used).

A12 Manipulate Files handles all remote requests to manipulate actual files. This includes printing local files, storing local files on the MSS, and requesting files from the MSS.

A13 Manipulate File Information handles remote activities involving information about files, but not the files themselves. This includes requests for MSS directory information, and updating and querying the catalog system.

A14 Manipulate Comms Links handles all remote requests to establish and terminate communication links to other remotes.

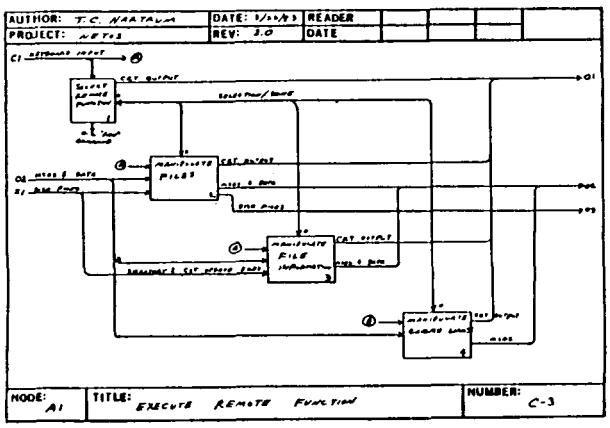

Figure 4: Example Structured Analysis Drawing.

SYSTEM 690 Approach

When considering an integrated environment, one can view integration from several perspectives. As shown in Figure 5, this includes integration of tools at the user-tool interface, integration between tools within any lifecycle phase, and integration across the entire lifecycle.

User level integration includes both consistency in interacting with the operating system (e.g. invoking tools via a menu) and consistency with interactive tool interfacing. This involves operating system specific issues as well as keyboard and display compatibility problems. It is planned to integrate the SDW environment with SYSTEM 690 to provide this level of integration, but this has not been done yet since in a heterogeneous environment a different version is required for each system.

Integration between tools is basically a question of compatibility of data, the ability of one tool to use the data generated by another tool. Compatibility itself can be viewed at a number of levels. In its most abstract form, compatibility of information is of concern. This is the biggest bar to integrating commercial tools from different vendors. Tools that use the same logical information may not have file format compatibility (a problem which frequently occurs when trying to integrate documentation from different word processors). Finally, two tools may not even have physical data format compatibility. Floppy disks written on one workstation may not be readable on a different one.

The tool-to-tool interface problem is being attacked at several levels. Compatibility of information is being controlled through the use of the data dictionary. File format problems are being handled through the use of a centralized database. The data dictionary definitions described earlier are decomposed into a set of third normal form relations which are maintained using the Ingres database management system. A data manager translates between the database and the file formats of specific tools. In order to minimize the amount of translation needed, a standard file format is used for all tools developed in-house.

Physical file format difficulties are avoided by using networking for all file transfers. This is available to any PC or workstation with a serial port using standard communication protocols, as well as between workstations and minicomputers via the Ethernet.

Integration over the lifecycle requires appropriate tools that are compatible with data used in two or more phases, and additional *mapping data* which relates items in the two phases. Currently there are no multi-phase tools in SYSTEM 690. It is anticipated that any such tools will be developed in-house, so that the mapping problem can be handled locally.

SYSTEM 690 Tools

Several classes of tools are in use or being developed for SYSTEM 690 which evolved in part from the SDW and other commercial tools. They include generic tools applied to the software engineering area, specialized graphical editors that allow creation or modification of data in a more graphical problem-oriented format, static analyzers that check various aspects of an existing design, and computer aided design (CAD) tools, including expert system and other AI techniques, that greatly assist the analyst or designer.

A number of old and new tools available for computers can be used to support the software development cycle. Some of the most useful are conventional text editors. Classically used for writing code and documentation, their big advantage is the universal availability and compatibility of text editors on all systems from micros to mainframes. By defining all standard file formats to contain only ASCII characters, a great amount of compatibility can be achieved in a heterogeneous environment. More sophisticated word processors are sometimes used to develop user's manuals, reports, and other such documentation. Here compatibility is maintained by defining a standard format (e.g. troff or TEX).

In terms of direct support for the SYSTEM 690 methodology, several tools have been developed to support data dictionary maintenance across all phases of the lifecycle. Although graphical tools to manipulate the data dictionary are being developed, they will be restricted to the more powerful graphics workstations. Therefore, we have developed a fill-in-the-blank forms editor for data dictionary entries that runs on a full range of computers [15]. The tool uses its own abbreviated ASCII files to store the data. Other data dictionary support tools include translators to convert between different file formats and the relational DBMS, and utilities for printing or viewing entries in the standard human-readable format [16].

To support the requirements analysis phase, an interactive structured analysis diagram editor is being developed on a SUN 3 workstation [17]. This tool makes it easy for an analyst to create and maintain such a diagram, while simultaneously updating the corresponding data dictionary entries. Similarly, an interactive structure chart editor is being developed to support the design and implementation phases.

Having all of the lifecycle data stored using a standard database manager makes it easy to develop static analyzers, tools that can examine the existing data dictionary information for consistency within a lifecycle phase and between phases. For the implementation phase there is also a style checker to analyze source code for adherence to local standards.

True computer-aided design tools are under development that will provide more than the ability to easily enter or examine design data. Several extensions are planned to the structured analysis diagram editor. These include the automatic placement of symbols and routing of lines, and software to help the analyst do the functional decomposition. An initial effort along these lines is a program that examines the data dictionary, and with interactive input from the designer analyzes the coupling and cohesion in a decomposition [12]. Similar extensions are planned to the structure chart editor. A planned extension to bridge the gap between the requirements analysis stage and the design stage will display a structured analysis diagram in

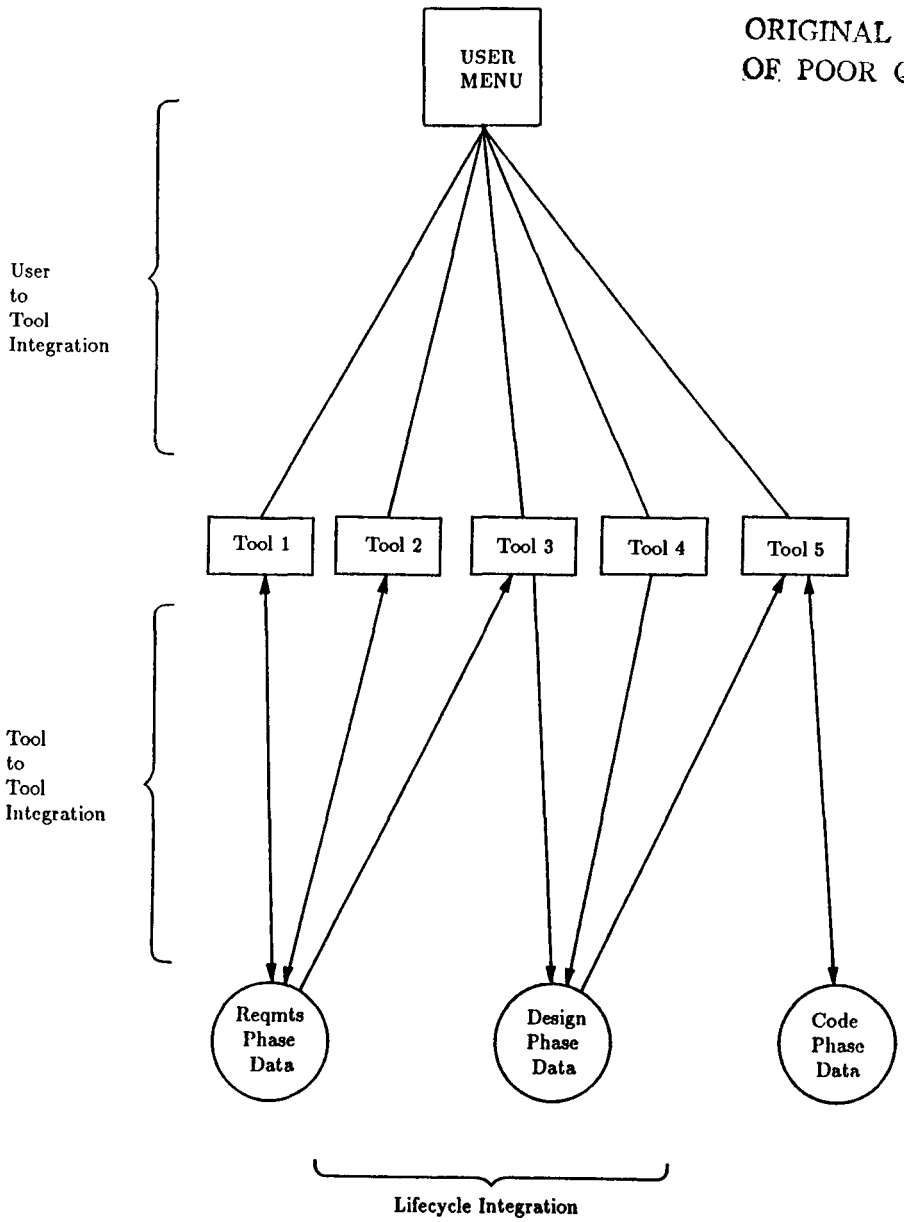

Figure 5: Types of Integration.

one window, and use transform and transaction analysis, along with expert systems techniques, to help the designer map directly into a structured design.

Testbed Considerations

A second objective of SYSTEM 690 is to provide a software engineering testbed to allow research into software engineering methodologies. The primary emphasis to date has been to develop ways of collecting data on the software engineering process. Tools are instrumented to collect usage and performance data to allow analysis of usage patterns [18] [15] [16]. The start and stop times of not only each tool, but of specific tool subfunctions, are recorded and stored in the Ingres database. A standard form for measuring user satisfaction with a given tool has been developed [19]. Standard statistical analysis packages are then used to analyze the data. A study is underway to determine what metrics are most needed to support the analysis of relative productivity for different software development methodologies.

Experiences and Plans

The first attempt at providing computer support was to create data dictionary entries in human-readable format using standard text editors and to store them in a common directory on a central VAX. Although this facilitated compatibility among the systems, the form of the files made it difficult to analyze or control data content with software tools.

The next version was a complete move to a centralized system. The data dictionary database was implemented under Ingres and an interactive editor was developed to run on the VAX that directly interfaced with the database. Although the database greatly simplified and encouraged the development of static analyzers and other tools, the load on the VAX from other applications slowed the editor response time to the point that users became frustrated. This experience with user dissatisfaction with response times made it clear that even PC level workstations are preferable for interactive tools. The development of the forms-based editor for the PC has been well accepted.

Most of our user experience has been in the design phase. The corresponding data dictionary has evolved with use and experience. We found several cases, mostly in the area of passed parameters, where what had been adequate for human understanding lacked the needed precision for machine readability. This required some augmentation of normal design techniques with rules of constraint to force a consistent and non-ambiguous design.

Finally, system reliability has turned out to be a critical issue. Although work can still be done when some components of the system are down, it is also true that there are more things that can go wrong. The communications network has been our biggest problem. Although the primary network is the 10 megabyte/sec Ethernet, the ability to use "uucp" or "kermit" over RS-232 backup links has proved essential. Critical items, such as laser printers and formatting software should be available on more than one machine.

The real future of the software engineering environment is in the use of graphic workstations coupled with AI techniques to create tools that truly aid the designer across all stages of the lifecycle. This requires a combination of interactive tools on heterogeneous workstations to provide a responsive user interface coupled with larger machines for more computationally intensive AI routines. Research issues include the determination of where AI can be applied in the design process and development of the corresponding expert knowledge, along with the development of techniques for integrating the heterogeneous environment in a manner transparent to the user.

The other primary research thrust planned is to utilize the instrumentation of the software engineering testbed to evaluate different software development methodologies, including rapid prototyping and object oriented design. In addition, investigation of tools and methodologies are needed for several specialized software development environments. These include VHDL, database design, AI systems, and parallel processing. Also being considered is the possible inclusion of other Computer-Aided Software Engineering (CASE) tools into the environments where source code is available. Examples include CASE packages from Textronix and McDonnell Douglas, DEASEL: an Expert System for Software Engineering (NASA) and the Software Engineering Testbed (Boeing/ Carnegie Group).

References

[1] Department of Defense. *Defense System Software Development.* Military Standard DOD-STD-2167, Department of Defense, Washington, D.C. 20301, June 1985.

[2] Steven M. Hadfield. *An Interactive and Automated Software Development Environment.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1982. AD-A210 920.

[3] L. Osterwell. Software environment research: directions for the next five years. *COMPUTER*, 14(4):35–43, April 1981.

[4] Robert L. Glass. Persistent software errors. *IEEE Trans. on Software Engineering*, SE-7(2), March 1981.

[5] A. Wasserman. Automated development environments. *COMPUTER*, 14(4):7–10, April 1981.

[6] Douglas T. Ross. Structured analysis (sa): a language for communicating ideas. *IEEE Transactions on Software Engineering*, SE-3(1):16–34, January 1977.

[7] *An Introduction to SADT: Structured Analysis and Design Technique.* Softech, Inc., Waltham, Mass., 1976.

[8] SofTech, Inc. *Integrated Computer-Aided Manufacturing (ICAM) Function Modeling Manual (IDEF0).* User's Manual UM 110231100, Materials Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson AFB, OH, June 1981.

[9] Kevin Rose. *Development of Interactive Computer Graphics Software System and Graphics Tools.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1982.

[10] R. Balzer, N. Goldman, and D. Wile. On the transactional implementation approach to programming. In *Proceedings, 2nd International Conf. on Software Engr.*, Long Beach, CA., 1976.

[11] Meilir Page-Jones. *The Practical Guide to Structured Systems Design.* Yourdan Press, New York, 1980.

[12] David W. Fautheree. *An Analysis Tool in a Knowledge Based Software Engineering Environment.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, March 1986. AD-A172 407.

[13] Stephen A. Wolfe. *A Natural Language Processor and Its Application to a Data Dictionary System.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1985. AD-A164 026.

[14] Charles W. Thomas. *An Automated/Interactive Software Engineering Tool to Generate Data Dictionaries.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1984. AD-A152 215.

[15] Jeffrey W. Foley. *Design of a Data Dictionary Editor in a Distributed Software Development Environment.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, June 1986. AD-Ax172 406.

[16] Charles W. Hamberger, Jr. *Analysis, Definition, and Implementation of a Network-based Microcomputer Software Development Environment for the AFIT Digital Engineering Laboratory.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, March 1986. AD-A172 781.

[17] James W.Urscheler. *Design of a Requirement Analysis Design Tool Integrated with a Data Dictionary in a Distributed Software Development Environment.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1986. AD-A177 663.

[18] Thomas C. Hartrum and Charles W. Hamberger, Jr. Development of a distributed data dictionary system for software development. In *Proc. of IEEE 1986 Nat'l Aerospace and Elec. Conf., Vol. 3*, pages 648–655, Dayton, OH, May 1986.

[19] Thomas C. Mallary. *Design of the Human-Computer Interface for a Computer Aided Design Tool for the Normalization of Relations.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1985. AD-A164 100.