Appendix F

UNIVERSITY OF COLORADO

(NASA-CR-163423) SOFTWARE DEVELOPMENT ENVIRONMENT, APPENDIX F (Colorado Univ. at Boulder.) 12 p HC A02/MF A01 CSCL 09B
N80-30067
Unclas
G3/61 28339

DEPARTMENT OF COMPUTER SCIENCE

Technical Report

Appendix F

UNIVERSITY OF COLORADO

DEPARTMENT OF COMPUTER SCIENCE

Technical Report

SOFTWARE DEVELOPMENT ENVIRONMENTS

William E. Riddle
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

CU-CS-182-80 October, 1980

This work was supported, in part, by grant NSG 1638 from NASA Langley Research Center.

RSSM/101
Prepared for a panel
at Compsac 80 Chicago,
(October 1980).

SOFTWARE DEVELOPMENT ENVIRONMENTS

William E. Riddle
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

Abstract: The purpose of this panel is to provide some assessment, based on experience, of the current status in the area of software development environments. This paper provides a context and vocabulary for the panelists' remarks by discussing the purposes of environments, the types of environments, the constituents of an environment, the issue of environment integration, and the problems which must be solved in preparing an environment. The paper also provides a focus for the panel by proposing some general maxims to guide near-term future work and by posing a number of questions which the panelists have been asked to address in their remarks.

The process of software development is slow, costly and error-prone. The problem seems to be that, unlike other creative disciplines, computer science lacks both a sufficient understanding of the properties of the products being developed and an adequate set of principles, practices and procedures for guiding the development process and reasoning about the software system as it is being developed. Even the most cursory comparison to analogous disciplines such as art, architecture, carpentry, etc., uncovers the immaturity of the discipline of software development.

Recently, solutions to this problem have been sought by preparing *software development environments* which provide facilities supporting the rational production of a software system's executable description. Such environments are intended to support not only the agglomeration and evolution of the system's description but also the assessment of its validity and quality and the exploration of alternative versions.

The purpose of this panel is to provide some assessment, based on experience, of the current status in the area of software development environments. This paper provides a context and vocabulary for the panelists' remarks by discussing the purposes of environments, the types

This work was supported, in part, by grant NSG 1638 from NASA Langley Research Center.

of environments, the constituents of an environment, the issue of environment integration, and the problems which must be solved in preparing an environment. The paper also provides a focus for the panel by proposing some general maxims to guide near-term future work and by posing a number of questions which the panelists have been asked to address in their remarks.

AIMS OF A DEVELOPMENT ENVIRONMENT

The primary purpose of a software development environment is to provide a context for the orderly, rational evolution of software systems. The premise is that aid in performing development activities is crucial to attaining an acceptable system, on schedule and with a reasonable consumption of resources; the approach is to support the use of development techniques and the observance of development principles during the development process; and the hope is that progress will be more steadily and quickly made.

A narrow definition of *software development* would encompass those activities which follow the definition of requirements and result in the delivery of an acceptable system. However, for two reasons we consider development to encompass *all* activities during the system's lifetime from its initial conception as a set of user or customer desires to its ultimate retirement. First, much of the support for development activities in the narrow sense are of value in performing other activities such as maintenance. Second, we feel that these other activities are not fundamentally different and that it is a mistake to consider them to be separable.

Software development environments which realize this central goal require a number of fundamental capabilities. Foremost is the capability to extract information about the system under development from the developers and organize this information into a coherent body of knowledge. This extraction and organization is obviously hard to do fully automatically, but something more is needed than simple facilities for recording information and interrelationships among pieces of information.

Also required is the capability to assess the consistency and implications of the recorded information. The assessment of consistency allows developers to gain confidence in the suitability and quality of their work — for example, if there are no contradictions between information describing a design and information describing the system's requirements then it is fairly safe to assume that the design is an appropriate one. The ability to assess the implications of the information provides the basis for making meaningful progress — by identifying decisions yet to be made and the information impacting and constraining the decisions — and allocating project resources.

To complement the capabilities to accumulate and assess information, the capability is required to retrieve information in a variety of forms. Partially, this is necessary because of the need to produce a large number of documents, for example, user manuals, code, maintenance manuals, design documentation, etc. Partially, it is necessary because developers need to be able to periodically regain a perspective of the system's characteristics.

The necessity of these general capabilities reflects the view that software system development, in the extended sense embraced here, is an information agglomeration task. The product at any point during development is a record of what is known about the system and the persons having legitimate interest in the system have a variety of needs for visibility into this record. The task at any point is to evolve the information either by adding new information reflecting new decisions or by eliminating internal inconsistencies existing because of incorrect decisions or because of changes to existing information. The purpose of a software development environment is, in essence, to facilitate, foster and rationalize the agglomeration of information.

CONSTITUENTS OF AN ENVIRONMENT

The facilities provided by a software development environment are delivered by *tools*. The more conspicious tools constituting an environment are computer programs, such as compilers, which augment the developers' "powers" in much the same way a saw augments a carpenter's powers. Equally important are intangible tools which are not implemented as computer programs but serve to guide and support the development process.

The constituent tools making up an environment may be divided into three classes. *Cognitive tools* extend the intellectual powers of the developers. They provide procedures, practices and principles which rationalize the development process itself and make it easier for the developers to solve the problem at hand. These tools come in the form of rules, guidelines and methods which focus the attention of the developers upon an appropriate succession of problems to be solved in order to achieve the development of a required system.

Tools in the second class, *notational tools*, extend the denotational powers of developers. They provide languages in which developers may record decisions and their effects. They allow the effective communication of information about the evolving system among the interested "agents" who are party to the development effort. They allow a record to be prepared which is an adequate basis for assessing the extent of progress and for identifying the suitability of what has been decided and the aspects of the system which have not yet been attacked.

The third class of tools are *augmentive tools* which extend the capabilities of the developers in much the same way in which saws, hammers and screwdrivers extend the capabilities of carpenters. Some of these tools perform the mundane tasks present in any development activity. Others perform those tasks which require an unfailing rigor and precision which humans, subject to fatique and prone to error, cannot always deliver. Still others carry out those tasks which even the smartest of humans find beyond their capabilities. All of the tools in this class serve to extend the reasoning, as opposed to the problem-solving, powers of the developers.

It is difficult to define tools within any one of these classes without also defining tools within the others. For example, focusing upon a specific development method, and thereby defining the cognitive tools which embody this method, has strong implications for the notational and augmentive tools which are compatible with the method and which serve to support it and each other. One must, therefore, co-evolve the various tools which are part of an environment.

TYPES OF ENVIRONMENTS

A wide range of different types of development environments are possible. One major factor differentiating these different types is the extent to which they focus upon a specific development method. One extreme is those development environments, called *toolboxes* or *toolkits*, which reflect no method in particular but rather provide help to developers no matter what "style" they use (including the use of no "style"!). The notational and augmentive tools in these minimally constrictive environments are typically minimally related and integrated. Further, the environment can usually be employed in the development of almost any type of software system.

At the other extreme are *development systems* which promulgate a single development method. The notational tools and augmentive tools in these types of environments are typically highly integrated and exhibit little, if any, redundancy. These environments are usually oriented towards the development of a single, rather restricted type of software system such as an accounting system or a non-linear equation solving system. These systems are akin to "plants" in the manufacturing world.

It is obviously desirable to have an environment which falls somewhere between these two extremes. Such a *development support system* would provide some guidance to developers in the form of a development "philosophy" or approach which helps them focus on the important issues at the various stages of development. Thus, in terms of the methods provided by a development support system, there would be a collection of complementary ones and the developers should assess the environment as "suggestive, helpful and supportive" rather than "imposing." Notational and augmentive tools should also foster this reaction and should present the developers with a homogenous, integrated context in which to carry out software development. It is probable that this type of environment will be oriented towards the development of a specific, but relatively broad, type of software system if only because this focus is necessary so that the environment is well-defined.

As one moves from toolboxes to development systems, one finds increased tool integration: duplication of tools decreases, tool specialization increases, and the tools complement each other to a greater degree. In an integrated collection of tools, the collection itself is more powerful than just the collective power of the constituent tools.

Notice that as the number of methodologies supported by an environment varies there is a concomitant variation in the variety and number of languages provided by the environment. Thus, another way to achieve integration of the constituents in an environment is to focus upon a small set of languages.

If done carefully, the result can be an environment that provides a set of highly integrated tools but that supports a wide range of methodologies.

OTHER ENVIRONMENT CHARACTERISTICS

In addition to classifying environments with respect to their "fixation" upon a development method, there are a number of characteristics which define other dimensions along which an environment may be classified. Some of the more important of these characteristics are the following:

- *agents serviced:* the environment will provide help to some subset of people concerned with the target systems, i.e., the systems prepared with the help of the development environment (customers, users, requirements definers, designers, programmers, acceptance testers, verifiers, trainers, managers, accountants, maintainers, modifiers, etc.);
- *characteristics of target systems:* the systems which can be developed with the aid of the environment will span some subset of data processing concepts (data management, numerical computation, text processing,real-time control, parallel processing, concurrent processing, distributed processing multiprogramming, etc.);
- *development phase:* the environment will be oriented toward some subset of the lifecycle of the target systems (requirements definition, system definition, architecture design, algorithm design, program implementation, system implementation);
- *properties of interest:* the environment will support the reasoning about the target system's functionality, performance and/or economics;
- *development task:* the environment will aid in performing some or all of the tasks which must be carried out during development (creation of a solution, certification of the solution, or exploration of alternative solutions);
- *development activity:* some or all of the activities performed by the agents concerned with the target systems will be supported (bookkeeping, supervision, management, decision-making, etc.).

These characteristics define a multi-dimensional space and a particular development environment will be oriented toward some subset of the possibilities along each dimension.

PROBLEMS IN THE PREPARATION OF DEVELOPMENT ENVIRONMENTS

To prepare a development environment a number of problems must be solved. Since an environment is itself a software system, these problems could be organized along the lines of the traditional life-cycle for software systems. It is more interesting, however, to group the problems somewhat differently from (although isomorphically to) the traditional life-cycle view since this different organization more clearly identifies alternative approaches to the preparation of an environment.

The problems which must be solved during the preparation of a development environment may be organized as follows:

- *scope problem:* the environment must be placed within the multi-dimensional space defined in the previous section;
- *capability problem:* the capabilities which the environment will provide (so that the various agents may perform their activities and tasks during the various phases in the development of the target systems) must be identified:
- *techniques problem:* techniques implementing the required capabilites must be developed (or identified and borrowed):
- *delivery problem:* the environment's organization must be determined, consideration must be taken of the computing facility on which the environment will execute, and the human-engineering aspects of the environment must be addressed;
- *training problem:* development practitioners must be educated in the use of the environment and helped to understand how to effectively use it (since the latter may necessitate approaching the development of systems in ways different from established practices);
- *evaluation problem:* the efficacy of the environment must be measured.

The statement of these problems is suggestive of one order in which they can be addressed. Obviously, a totally sequential solution of these problems in the order given is not possible since the problems are highly interrelated. The evaluation problem, for example, requires that some consideration be given to it during the solution of the delivery problem so that the necessary information for evaluation may be extracted during the use of the environment. Nonetheless, attacking these problems in the order given is a reasonable approach. It necessitates, however, a subscription to an overall approach to preparation which is heavily exploratory in nature. This is not at all unusual for systems, such as environments, for which there are no prototypical implementations.

SOME DESIRABLE ATTRIBUTES

Regardless of the type of a software development environment and its characteristics, there are a number of general attributes which seem appropriate at this point. Perhaps these attributes will be shown undesirable once we have more experience in designing, implementing and using software development environments; but they do seem to provide some general guidelines for preparing development environments in the near future.

Tools should be "toys." Tools should be as simple as possible and should serve a narrowly-defined purpose. The temptation to produce multi-purpose, general tools should be resisted in favor of producing single-purpose, specific tools.

Tools should be independently available. Collections of tools should have a user interface which allows the individual tools to be used independently or sets of tools to be used in any (reasonable) sequence. The tendency, evidenced by some programming language-specific development environments, to hide individual tools (e.g., the parser) should be avoided.

Tools should be catalogued. In addition to their function, tools have many characteristics which are important to know when deciding whether or not to use them. Some of these characteristics are: assumed user sophistication, assumed development method, types of systems being built, and user development strategy.

Tools should be methodology independent. A development method, i.e., an overall appraoch to the development process, can be rigorously defined as a set of rules for how to use a collection of tools to carry out the development process. The best tools are ones which can be used to support a variety of methodologies and therefore themselves minimally impose constraints upon their useage.

Notational tools should be numerous and simple. Many things must be described about a system during its development, and it is better to have a number of compatible languages, each with a single purpose, than to put everything into one language.

Notational tools should allow modelling. Each language should provide abstraction capabilities so that it is possible to not only highlight specific aspects but specify these aspects to an appropriate level of detail.

The notational tools' semantics should be developed first. Each language should admit rigorously defined models. The formal foundation for the language should be carefully and fully developed prior to determining a user-friendly syntax for describing models based on this foundation.

"Old programmers never die, they just become designers." Therefore, languages for describing operational aspects of a system should have their heritage in programming languages. Further, languages for describing non-operational aspects (e.g., behavior) should *not* use concepts from programming languages.

Augmentive tools should be exceptionally robust. My daughters have an unbelievable knack for finding new, innovative and creative ways to use the hammers and screwdrivers in our toolbox. This can be expected to be the situation with software development tool users also and so the tools should be able to withstand unusual, even bizarre, use. They should also protect the user from doing (too much) harm.

Augmentive tools should deliver feedback. It is notoriously difficult to algorithmically assess the suitability of a system. Tools to aid this assessment should therefore do what *can* be done and provide developers with algorithmically derivable information which the developer can interpret in order to determine suitability.

The premier augmentive tool is a data base. The core of a collection of tools should be a data base which retains and organizes all information known about the system under development. The data base should allow inconsistent pieces of information to co-exist. All other tools should access the information through the data base management routines.

EXPERIENCES WITH EXISTING ENVIRONMENTS

A number of software development environments have been prepared and used. To complement the general discussion of environments presented in this paper, reports on experiences with some of these environments will be presented by the panelists. These reports will be given by:

G. Estrin, University of California at Los Angeles
W. Riddle, University of Colorado
S. Saib, General Research Corporation
P. Santoni, Naval Ocean Systems Center
A. Wasserman, University of California at San Francisco

In their reports, these people have been asked to focus on the following questions:

1. What is the type of your environment and what tools are provided?
2. How is the environment organized and to what degree is the collection of tools integrated?
3. Where does the environment sit in the multi-dimensional space defined above?
4. To what extent and how have the problems in preparing software development environments been solved?
5. What were the technical and non-technical influences which shaped the nature of the environment?
6. What were the technical, political, and pragmatic problems encountered in preparing the environment?
7. Do your experiences reinforce or negate the maxims given above for guiding near-term future work on environments?
8. What are the important issues and/or problems to be solved in the future in order to provide truly effective development environments?