A SYSTEMATIC LOOK AT PROTOTYPING

Christiane Floyd
Institut für Angewandte Informatik
TU Berlin Sekr. SWT FR5-6

Franklinstr. 28/29
1000 Berlin 10
West-Germany

Contributors:

M. Kersten, K.D. Kreplin,
L. Mathiassen, H.C. Mayr,
P. Schnupp, J.M. Sykes, K.-H. Sylla,
H. Wedekind, H. Züllighoven

Contents

1. Introduction

This paper originates from a series of discussions between programme committee members during the preparation of the Working Conference on Prototyping. While trying to define the topic of the conference, it became clear to us that we each held our own viewpoint on the subject. Views differed as to the specific use of terminology as well as the application-oriented emphasis on particular strategies, and so did our judgements about the potential usefulness of prototyping. The views did not, however, seem contradictory but rather complementary.

It was therefore decided that there should be an introduction to the conference which would provide a unified conceptual framework for the different views. Since I was fascinated by this idea, I volunteered as chief author. However, the text draws extensively on position papers submitted by the contributors mentioned above which were available to me while writing the paper. It has also benefitted greatly by the constructive criticism of several of these authors. Some contributors have presented their own work on prototyping at the same conference (see [BUDDE, CHRISTENSEN, LEIBRANDT, MAYR] or elsewhere [FLOYD, STREICH]).

The basic difficulty in trying to define "prototyping" in connection with software development is that the concept cannot be used in its strict sense. This has also been recognized in other work on prototyping (see for example [MASON, SMITH, SQUIRES]). A "prototype" literally means "first of a type", a notion which makes sense in those branches of engineering where the manufacturer's aim is to mass-produce goods of the same type. Prototyping, here, refers to a well-defined phase in the production process, where a model is produced in advance, exhibiting all the essential features of the final product, for use as test specimen and guide for further production.

Software development differs from this kind of manufacture in several ways: it takes place in the context of an overall system development process; this development process normally leads to one single product and the desired characteristics of this product are not well-defined in advance. Thus, no relevant notion of "type" is available for software, and it is not clear what is meant by "first", i.e. how the prototype should relate to the final product. The situation is further complicated by the fact that the notion of prototyping can be meaningfully applied both to the desired product as a whole and to parts of it (for example, one might build the prototype of a car engine). And lastly, our use of the term "prototyping" in connection with software development indicates that we are primarily interested in a process rather than in the "prototype" as a product. What we are looking for, then, are processes which involve an early practical demonstration of relevant parts of the desired software on a computer, and which are to be combined with other processes in system development with a view to improving the quality of the target system.

Many software developers are motivated to employ prototyping by important conclusions drawn from their working experience:

1 You only know how to build the system when you have built it - and then it is often too late.

2 When developing software for your own needs, you often build version by version; while working with the tool, you get good ideas for the facilities it should provide to suit the job in hand.

In order to make constructive use of these experiences, a prototype should always be considered a learning vehicle providing more precise ideas about what the target system should be like. The overriding concern in prototyping is a commitment to the quality of the desired final product.

In this paper, we consider prototyping as a component of software development methodology. In contrast to the phase-oriented approach, which assumes a strictly linear ordering of development steps, prototyping serves to introduce an element of communication and feedback. As shown in section 3, different approaches to prototyping vary in the degree to which this linear ordering is broken down and mapped onto successive development cycles.

Though much of what follows can be generalized , the discussion focuses on software intended as a direct support for human work. The class of software systems considered here comprises, for example, software as part of information systems, text processing systems and programming environments. Such software systems tend to be characterized by three aspects: an interactive user interface, algorithmic expert knowledge, and a large data base. For lack of a better name, we shall refer to them as interactive application systems throughout this paper.

The quality of interactive application systems is largely determined by their adequacy as tools for human users. This involves both psychological and social factors (for example human cognition, communication and co-operation) which cannot easily be discussed on the basis of defining documents. Therefore, prototyping for interactive application systems normally serves to enhance the communication between developers and users concerning the suitability of man-machine interfaces or, more generally, the correspondance between system functions and work tasks.

This is, however, not the only point of concern. We may also be interested in other aspects such as demonstrating the technical feasibility of a software idea or determining the efficiency of a part of the system. In such cases, it may be appropriate to use prototyping within the development team, for example to aid communication to design and specification or to help structure the implementation of a large system.

In what follows, first, the process of prototying is characterized as consisting of four steps, which must be considered together. Then, several approaches to prototyping are distinguished on the basis of the goals one wishes to achieve. Here, the aim is to cover the whole spectrum of processes commonly denoted as "prototyping", even if in some cases the term seems ill-chosen. Prototyping strategies, pertaining to the various approaches are described, which serve to tailor the prototyping process to a given development situation. Next, techniques and tools for prototyping are presented, which support construction and further use of prototypes; requirements regarding the tools are derived from the different approaches to prototyping. Last, prototyping is placed in the context of system development as a whole and related to the interests of the people involved: developers, customers, users.

2. Characterization of Prototyping

Prototyping can be seen as consisting of four steps: functional selection, construction, evaluation, and further use. This section describes what the steps are and how they relate to each other.

FUNCTIONAL SELECTION refers to the choice of functions which the prototype should exhibit. The selection should always be based on relevant work tasks which can serve as model cases for demonstration. The range of features offered by the prototype is never the same as that of the final product; otherwise they would be identical. The difference between the functional scope of the prototype and the product may be that

- the system functions implemented are offered in their intended final form but only selected functions are included ("vertical prototyping");
- the functions are not implemented in detail as required in the final system; thus, they can be used for demonstration, part of their effect being omitted or simulated ("horizontal prototyping").

Often these two elements are combined in different parts of one prototype.

CONSTRUCTION refers to the effort required to make the prototype available. Generally, this effort should be much smaller than that involved in developing the final product. This can be achieved both by appropriate functional selection and by the use of suitable techniques and tools for prototype construction.

In constructing a prototype, the emphasis should be on the intended evaluation, not on regular, long-term use. This implies that certain quality requirements pertaining to the final product, such as reliability, data security or efficiency, can be disregarded unless they are a part of what the prototype is supposed to demonstrate. If, for example, the prototype is to be used for finding out whether the response time of the system is suitable for the desired work rhythm, then efficiency is an essential aspect and cannot be neglected. Similarly, in a banking system data security may be of paramount importance.

EVALUATION must be considered to be the decisive step in prototyping since it provides feedback for the further development process. Therefore, the overall project organization must ensure that resources are made available for the evaluation, including the participation of all relevant groups of prospective users. The evaluation only makes sense after appropriate training. It should be based on a document making explicit the criteria for the evaluation and specifying the work steps to be performed with the system. It is essential that the context of these work steps in the surrounding organization be taken into account and that attention also be paid to aspects of the work which cannot be formalized such as time, place, interuptions, communication with others.

Evaluation may take place both of the level of single users working with the system, in which case the emphasis is normally on cognitive problems involving the man-machine interface, or it may pertain to the co-operation between several users or between users and other people, in which case problems of communication between people need also be considered.

There are several possibilities for the FURTHER USE of the prototype. Depending on the experiences gained with the prototype and on the available production environment, it may merely serve as a learning vehicle and be thrown away afterwards, or it may be used fully or partially as a component of the target system. This is discussed in more detail in section 3.

Since the prototype is to be used primarily as a learning vehicle, care should be taken to design the prototyping process as a learning process.
This involves several aspects:

EARLY AVAILABILITY: A prototype should be available very early in software development so as to offer full benefit to all parties concerned: developers, customers, and users (hence the term "rapid prototyping").

DEMONSTRATION, EVALUATION AND MODIFICATION: A prototype should work in such a way that it can be demonstrated to the users. The demonstration should make sense in the context of the users' work processes, i.e. it should involve authentic and nontrivial problems so as to make the evaluation relevant. There should be an easy way of changing the prototype by revising existing or adding new features as needed in the evaluation, so as to allow modification cycles on the basis of one prototype.

TEACHING AND TRAINING: After evaluation and modification, a successful prototype may serve as a teaching environment to prepare prospective users for their work with the target system.

COMMITMENT: It must be kept in mind that if a prototype is demonstrated and there is a discussion with the prospective users about its evaluation, the commitment to the target system is very strong. Should essential changes of some features of the prototype be made during implementation of the final product without the explicit consent of the user, serious problems regarding its acceptance must be expected.

Exactly what kind of commitment is relevant depends on the specific approach to prototyping. This will be discussed in the next section.

3. Approaches to Prototyping

Depending on the goals we wish to achieve, we can distinguish three broad classes of prototyping:

- prototyping for exploration, where the emphasis is on clarifying requirements and desirable features of the target system and where alternative possibilities for solutions are discussed,
- prototyping for experimentation, where the emphasis is on determining the adequacy of a proposed solution before investing in large-scale implementation of the target system,
- prototyping for evolution, where the emphasis is on adapting the system gradually to changing requirements, which cannot reliably be determined in one early phase.

"Exploration" and "experimentation" are not used here in the strict technical sense these terms have acquired in the social sciences. This means, in particular, that the experiments involved are soft, since they cannot, at present, be based on a rigorous theory.

Admittedly, the borderline between exploration and experimentation is fuzzy. Also, prototyping for evolution may contain the other two kinds as part of an overall strategy. Nevertheless, the distinction is useful for clarifying the relation between prototyping and the system development process as a whole. For each approach, the intended relation between the prototype and the target system, the compatibility with the well-known phase-oriented model of software production, and implications for techniques and tools will be discussed.

3.1 Exploratory Prototyping

This approach focuses on the basic problems of communication between software developers and prospective users, particularly in the early stages of software development. The developers normally have too little knowledge about the application field, while the users have no clear idea of what the computer might do for them. In this situation, a practical demonstration of possible system functions serves as a catalyst to elicit good ideas and to promote a creative co-operation between all parties involved. Such a demonstration should not focus exclusively on one particular proposed solution, but should point out alternatives whose respective merits can then be discussed.

Since exploratory prototyping is very informal by nature, one cannot expect any detailed rules on how to proceed. In particular, the steps: functional selection, construction, and evaluation, take place as needed within the overall communication.

For exploratory prototyping to be successful, the developers must be aware that in spite of the informality of the process the users' expectations will be deeply influenced by the exposure to the prototype. Therefore, even in exploratory prototyping, there should be a strategy pertaining to the choice of features to be demonstrated and their relation to each other. For example, it might be useful to start with a set of functions permitting the users to perform one of their work tasks completely with the help of the prototype. They can then assess the suitability of the proposed partial solution for the particular task in hand, and also generalize by drawing analogies or pointing out dissimilarities to other work tasks.

On the other hand, it must be clear to the users that there is no commitment to reproducing the prototype in the target system, but rather to incorporating the good ideas derived from the exploration in its definitive specification. In order to avoid serious misunderstandings, it is of paramount importance to make explicit exactly what the users like or dislike about a proposed feature, and come to a common understanding on how this should be taken into account in the target system.

Exploratory prototyping is compatible with a phase-oriented approach to software development, and serves to enhance the early phases: requirements and functional analysis. The prototype can be considered as an aid to establishing what features the target system should offer. These are subsequently codified in the systems specification.

An exploratory prototype comes into being gradually as a result of a communication process; hence, it may generally be expected to be messy and unstructured, and is normally thrown away. Since it has no well-defined relation to the final product, the tools used to produce it need not be integrated with the production environment of the target system. For example, the prototype might be programmed in a high-level language permitting ease of expression and rapid local change, but offering no suitable concepts for decomposing large programs. Exploratory prototyping can only be recommended if there are tools available which keep to a minimum the effort required in constructing the prototype, and if the expected life time of the system is long enough or the quality requirements are sufficiently high to warrant the extra investment.

3.2 Experimental Prototyping

In this approach to prototyping a proposed solution to the customer's problem, is evaluated by experimental use. There may be various areas of concern, including the transparency of the man-machine interface, the acceptability of the intended system performance, or the feasibility of a proposed solution with the available resources.

Also, the nature of the experiment and the strategy chosen to carry it out may vary considerably. Since they influence deeply the communication with the users and the quality of the feedback obtained, they ought to be carefully agreed upon in advance by developers and users.

Experimental prototyping is closest to the original meaning of the term "prototyping". Even so, it would be more useful to distinguish several different cases on the basis of the strategy chosen.

FULL FUNCTIONAL SIMULATION is based on a prototype exhibiting all the functions of the target system intended to be available to the users for normal use. The prototype may be constructed by techniques which offer ease of implementation and modification rather than efficiency of the resulting system. Such a system may well be impossible to use as a production system owing to lack of efficiency or the absence of facilities for handling errors or special cases.

HUMAN INTERFACE (FRONT END) SIMULATION presents the users with the proposed man-machine interface, for example in the form of a dialogue, in its intended final form, but uses a mock-up for all other parts of the system. Thus, the users see what looks like a real system, but there is usually little validation of the input, and there may be no real data behind the program at all.

SKELETON PROGRAMMING exposes the users to the overall structure of the system on the basis of a few system functions selected as being relevant. This involves the design of the whole system and a drastic reduction of its implemented functional scope. The functions implemented in the prototype must enable the users to perform some of their work tasks fully and draw analogies to other work steps which are not supported by the prototype. This form of prototype is of interest for demonstrating how the system will be embedded into the users' overall work processes and for showing the interplay of manual and computer-supported work steps. It can also be used to simulate the intended target system's efficiency and thus get a feeling of what would be acceptable to the users.

BASE MACHINE CONSTRUCTION: this form of prototyping is on the borderline between exploratory and experimental prototyping. It involves the implementation of basic or primitive functions intended to be available to the users, plus the facility of combining these primitive functions with higher-level functions as desired by the users during the evaluation. It is particularly relevant when the software is to be used by several user communities whose needs do not exactly coincide.

PARTIAL FUNCTIONAL SIMULATION is used purely to test a hypothesis about the system. For example, to see whether a proposed algorithm will produce acceptable results in a worthwile proportion of cases using a reasonable amount of resources.

There is no universal answer as to which of these forms of prototyping is best and whether there should be one or several successive prototypes. The choice of a prototyping strategy must take into account the particular communication needs of the situation in hand as well as the available resources, techniques and tools. Even amongst the authors contributing to this paper there is no common judgement in this area. Some argue, for example, that full functional simulation is the best, or even the only proper, form of prototyping, whereas others point to the difficulties of a controlled evaluation of the whole system with reasonable effort. There is also the danger that the prototype, even if not found really suitable, might be retained as a production system owing to the unwillingness to spend further funds on a high-quality system. In this case, the net effect of prototyping would be contrary to the intention of using the prototype as a learning vehicle in order to achieve higher quality.

In phase-oriented software production, experimental prototyping is appropriate in any phase after the initial specification has been written. Some forms of experimental prototyping (full functional simulation and human interface simulation) are geared to the needs of communication between developers and users only. Others, for example skeleton programming and partial functional simulation, are also relevant for aiding communication within the developers' team and helping structure the implementation of large systems.

An experimental prototype should always be regarded as an enhancement of the target system's specification. Depending on the phase in which prototyping takes place, the prototype could serve as

- a complementary form of specification
- a form of refinement of (parts of) the specification
- an intermediate step between specification and implementation.

The possibilities of further use of a successful prototype after evalutation depend both on the strategy chosen and the tools available.

If the production environment of the prototype is not integrated with that of the final product, the prototype is thrown away. Therefore, certain prototyping strategies, in particular full functional simulation leading to a complex prototype, depend on the availablity of tools which minimize the effort of prototype construction (as in exploratory prototyping). Alternatively, the prototype may be used as it is as (part of) the final product, or perhaps it may be incorporated in a revised form, for example on the basis of new requirements which were clarified in the evaluation. This is particularly relevant in connection with human interface simulation and skeleton programming. These stategies for partial prototyping are also well suited as early steps in an evolutionary approach. They are, at least in principle, independent of special prototyping tools. If they are based on such tools, the production environment for the prototype must permit some kind of computer-aided transition from the prototype to the product. A last possibility is the automatic transformation of the prototype, or of parts of it, into (parts of) the product. This is highly desirable for all forms of prototyping, but requires tools which make such a transformation possible. Currently, systems for transformating an executable specification into efficient code, for example, are not commonly available to the practitioner.

3.3 Evolutionary Prototyping

This approach to prototyping is, one the one hand, the most powerful, on the other hand, the most remote from the original meaning of the term "prototyping". In the eyes of some authors, it should not be called prototyping at all, but rather development in versions ("versioning"). We include it in this classification because the borderlines between evolutionary and other approaches to prototyping are not sharp (any prototyping permits the revision of requirements and therefore contributes to the evolution of interactive application systems), and because exploratory and experimental prototyping are appropriate early steps of an evolutionary strategy. Evolutionary prototyping is based on the experience that

- the organization surrounding the interactive application system evolves, and therefore new requirements emerge
- the interactive application system itself, once it is used, tansforms its usage context and thus itself gives rise to new requirements.

In order to master these difficulties, a prototyping strategy primarily geared to overcoming communication problems, but otherwise sharing the assumptions of the phase-oriented software production model, i.e. that one software system with fixed requirements is to be produced, does not suffice. Instead, we need a dynamic strategy which views the product itself as a sequence of versions, so that each version can be evaluated and serves as a prototype for its successor.

Evolutionary prototyping always breaks down the linear ordering of development steps and maps it onto successive development cycles. Depending on the degree to which this takes place, we can distinguish between the following forms of development:

INCREMENTAL SYSTEM DEVELOPMENT, also called "slowly growing systems". The underlying idea is that complex problems are dealt with by a stepwise extension of the solution. Thus, for example, existing concepts or strategies for information processing are taken as a basis, and only a few elements are substituted by new computer-supported components. Against the background of a long-range development strategy, the design of the technical environments is accomplished gradually in a process of learning and growth. In order to be successful, the stepwise extension of the solution should be carefully geared to the work tasks to be supported by the system. In this way, it becomes possible to gradually train and involve users in the system development process with obvious benefits to the communication between users and developers.

Incremental system development is still fairly compatible with the phase-oriented software production model, in that it primarily affects the implementation phase, but is based on one overall design.

EVOLUTIONARY SYSTEM DEVELOPMENT views software development as a whole as a sequence of cycles: (re-)design, (re-)implementation, (re-)evaluation. In this approach, all phases are mapped onto successive development cycles. The emphasis is put on software constructed within a dynamic and changing environment. So, instead of trying to capture a complete set of requirements in advance, the system is built so as to accommodate subsequent, even unpredictable, changes. This approach takes into account the harsh reality of system development in a changing world. In order to work, it requires on the part of both developers and users a willingness to open themselves to communication and change to a high degree. In particular, developers must accept the necessity of revising their own programs repeatedly, which implies the need for a very disciplined work style. Users, on the other hand, must be willing to accept repeated system changes affecting their work and requiring new learning processes.

The number of cycles involved (and therefore the degree of deviation from the phase-oriented approach) can be tailored to the needs of the situation in hand. For example, there can be one or two cycles of exploratory prototyping during requirements analysis, one or more experimental prototypes during design, and incremental system development during implementation. In evolutionary system development, there is no maintenance phase as such; it is replaced by further development cycles based on the existing system version and new requirements.

In the case of evolutionary prototyping, the transition between prototype and product becomes mandatory at some stage; this means that, apart from possibly separate steps of exploratory and experimental prototyping at the beginnig, the production environments for prototype and product must be integrated. If there are no special tools available, evolutionary system development can be used within the regular production environment. It is greatly facilitated by tools normally associated with program maintenance: sophisticated text editors, project support libraries, project data bases supporting version control.

4. Techniques and Tools for Prototyping

As has already been suggested in several places above, prototyping does not entirely depend on new techniques and tools. Rather, it can be achieved to a large extent by the intelligent use of existing ones.

Perhaps the most important techniques relevant in connection with prototyping are: modular design, dialogue design, and simulation.

MODULAR DESIGN is of great importance in all prototyping strategies which aim at incorporating the prototype into the target system. These strategies include human interface simulation, skeleton programming, and incremental system development. In all these cases, modularity is a great help in facilitating the gradual replacement of prototype components by target system components. Interfaces cannot generally be expected to remain unchanged in the course of such a replacement because revised requirements may have a profound influence even on the whole system's structure. However, an explicit communication between modules via named interfaces makes the transition more reliable.

DIALOGUE DESIGN serves to make user interfaces transparent and flexible. For the evaluation of a user interface to make sense, it must be possible to name, discuss and change aspects of the dialogue at various levels of detail. These aspects include the overall structure of the dialogue, the choice of system commands, the layout of screens, the handling of errors and special cases, and the possiblities provided for structuring the work with the system as needed. Care should be taken to disentangle as much as possible conversational aspects of the dialogue from application-specific processing aspects in order to be able to change either of them without undue side-effects on the other. Such conversational independence, to the extent that it can be obtained, will greatly enhance the flexibility of interactive application system design.

SIMULATION is a widely used term with several meanings. Every program is a simulation of an activity happening in the real world. Also, traditionally, application cases are simulated in order to validate dp systems with respect to through-put, bottlenecks or reliability. As a technique for prototyping, simulation serves to make available those parts of the target system which, on the one hand, are not supposed to be demonstrated in their intended final from, but, on the other hand, cannot be omitted entirely in a realistic evaluation.

Typical candidates for simulation in connection with prototyping are file management and organization, which in the target system need to be based on careful analysis of storage capacity and retrieval needs, but in some prototyping strategies can be simulated by a trivial in-core data organization, producing a small number of realistic and relevant test cases. Other forms of prototyping involve the simulation of the response time of the target system with the help of simple loops. It might also be useful to simulate those parts of a user interface which are not needed in connection with a particular work task.

Used as a technique, simulation is clearly dependet on modular design, which permits simulation to be made locally without affecting the system structure as a whole.

As far as TOOLS are concerned, prototyping does not require the re-invention of the wheel. Since prototyping became popular, several existing methods and tools have been proposed under this heading. They include very high-level languages, data base management systems, dialogue definition systems, interpretive specification languages, and symbolic execution systems.

All of these support the rapid construction of operational system versions which can be demontrated. On the other hand, they are all general-purpose tools offering no model of the application area and no building blocks for the target system. In addition, they often lack flexible interfaces and have not been designed with a view to integration of the results derived with the help of these tools.

One step beyond these general-purpose tools, one may find application generators, program generators and collections of reusable software. Whereas such individual tools are often confined to very restricted application fields and mostly involve very simple and well-formalized application models, a well-assorted library of reusable software with suitable interfaces can be used to construct large parts of an interactive application system with little effort.

The most promising computer-aided support seems to be a programming environment offering a well-integrated set of tailored tools for prototype and target system construction with facilities for computer-aided transition or even automatic transformation between the two. In contrast to many existing programming environments, which enforce some directive approach to software development, based, for example, on a top-down, deductive design methodology, a programming environment suitable for prototyping would support a design methodology that emphasizes creativity, experimentation, learning, and evolution. Unfortunately, at present, such systems are still in the realms of research.

As a more realistic short-term alternative, prototyping tool-kits may be used, which may be tailored to specific application areas (for example, interactive application systems).

A tool-kit should comprise an editor and a text processing system, a high-level language, a data base system, simulation facilities, a statistical package, a window/screen management system, and a compiler-writing system. In order to make use of existing software, a program library is needed as well as tools for configuration, embedment, and integration of existing programs.

All tools need to be integrated to make effective work possible. Tools dealing with man-machine dialogue should be interactive and self-guiding so as to allow for changes as needed while evaluating the prototype.

Routine information should be gathered automatically by the tools and stored in a data base so as to be readily available in subsequent work. Detailed documentation should be done as far as possible by the tools in the tool-kit and should require human intervention only to determine the form and extent of the detailed design documents.

5. The Impact of Prototyping: Dangers and Potential Benefits.

As has been pointed out, prototyping serves to introduce into software development methodology an element of communication and feedback. Prototyping relies on strategies that carefully take into account several aspects of the situation in hand, such as specific communication needs, work capacity, and available resources.

Prototyping is not, in itself, a method for system development. It does not prescribe a sequence of steps which guarantee that an operational system satisfying all requirements is derived from fuzzy user concepts and attitudes. It should rather be considered as one procedure within system development that needs to be combined with others. Each approach to prototyping is bound to a specific development strategy (for example, a phase-oriented or a cyclic approach). This overall strategy will have more impact on the outcome of software development than the prototyping procedure in itself.

Prototyping offers the means to demonstrate an operational version of the target system early on and to find out from one or several such versions what developers can do and what users want to have done. It does not, in itself, offer any guarantee for real user participation or for considering the interests of those affected by the development process. Like any other procedure, it may be misused to manipulate users into co-operating in what amounts to job obsolescence or the downgrading of their own work. By presenting prospective users with new technical gadgets "to play with", prototyping may conceal the fact that this type of user involvement can be a sophisticated way of tuning the system contrary to the users' interest.

On the other hand, if used in the framework of an appropriate development strategy, prototyping can contribute significantly to the development of more adequate systems. For example, in conventional software development, requirements analysis is usually cut back to a minimum. After an overall business analysis, the fact-finding analysis and the identification of needs and interests are done merely in passing. Since there is no method available for modeling the dp system in relation to the application field, the mapping process from the model of the application field, described in the requirements specification, to the software system design is done intuitively and from an exclusively dp-oriented viewpoint. So, there is a substantial risk that the first prototype is fundamentally inadequate (it incorporates an unsuitable model), and that all discussions and subsequent revisions of this prototype concentrate on peripheral symptoms.

Therefore, a combination of analytic and experimental activities seems to be more appropriate. Such an approach to the development of interactive application systems might be summarized as follows: after an overall business analysis ("what do we want to change and why?"), analyze the working situation and develop a model of it in terms familiar to the users; then try to elaborate the requirements specification and the design for a potential software system using the basic elements of the application field model; based on this system model, construct a prototype and let it be evaluated by the users; repeat this cycle until the necessary revisions at the end of a cycle indicate a near stable set of requirements; then build the actual target system incrementally, using the last prototype as part of its specification.

The phase-oriented approach, if it works, makes control of the development process easy. By contrast, the exploration, experimentation and evolution involved in prototyping are somehow not suited for external and prior control. They require a willingness to accept criticism and changes from developers and users alike. They also rely on a sense of partnership which can only arise in a development setting where users' interests are taken seriously. There is a basic contradiction here between efficiency of the process and quality of the product. High demands are made on both developers and users to make prototyping successful.

As a closing remark, it should be pointed out that prototyping does not necessarily increase the overall costs of software development, since the extra investment for the prototype helps to reduce the probability of costly, fundamental misunderstandings. Prototyping substitutes the anticipation of a future system by a process of learning and practical experience. It is important to realize that prototyping aims at the technical kernel of a development process only. It does not automatically solve the problems of social and organizational implementation. The technical process of prototyping is not yet completely understood. This holds for supporting tools as well as for the concepts of selecting and constructing prototypes. Even so, because of its potential benefits prototyping should be seriously considered in all system development.

6. Afterthoughts

At the end of the conference a few additional comments seem in order. The characterization of prototyping approaches presented in this paper has been generally considered helpful in clarifying the discussions amongst the participants.

I would like to mention some divergent proposals for a classification as suggested by other authors: one was to use the term "prototype" only for such early operational system versions which are intended to be thrown away, and otherwise to use the term "pilot system" as it is done in chemical engineering (see [RZEVSKI]); another author suggested the term "model" for all early versions which are intended to support the design process only and to reserve the term "prototype" for such versions which are intended for actual, though preliminary use (this was referred to as "organizational prototyping", see [TAVOLATO]).

Whereas traditionally the term "prototype" is used as a noun, Bill Riddle suggested to use it as an adjective and in this paper it is mainly used as a verb.

All of these terminologies have their merits in specific contexts and should be used wherever appropriate, as long as it is clear what concretely is implied. By contrast, there seems to be no established context-free terminology available. Therefore, Peter Naur in his final comment suggested to drop the term "prototyping" altogether, as it does not seem helpful in discussing questions of software methodology. I should like to point out that the substance of the present paper would not be affected if the term "prototyping" were avoided throughout the text.

Acknowledgement

In addition to the contributors mentioned in the introduction, I would like to thanl the following persons for their helpful criticism of the preliminary version of thi: paper: Stefan Jähnichen, Wilfried Koch, Peter Naur, Erhard Nullmeier, Christoph Oeters, Michaela Reisin and Karl-Heinz Rödiger.

My special thanks go to Philip Bacon who has done a superb job of polishing up th final text.

References

BUDDE, REINHARD, AND KARL-HEINZ SYLLA:
From Application Domain Modeling to Target System.
(In this volume).

CHRISTENSEN, NIELS, AND KLAUS-DIETER KREPLIN:
Prototyping of User Interfaces.
(In this volume).

FLOYD, C.:
On the Use of "Prototyping" in Software Development.
HP3000, International Users Group Meeting 1981.

LEIBRANDT, UTE, AND PETER SCHNUPP:
An Evaluation of Prolog as a Prototyping System.
(In this volume).

MASON, R. E. F., AND T. T. CAREY:
Prototyping Interactive Information Systems.
CACM. VOL.26,5, 347-356 (1983).

MAYR, H. C., M. BEVER, AND P. C. LOCKEMANN:
Prototyping Interactive Application Systems.
(In this volume).

RZEVSKI, G.:
Prototypes versus Pilot Systems: Strategies for Evolutionary Information System Development.
(In this volume).

SMITH, D. A.:
Rapid Software Prototyping.
PhD thesis, ICS Dept., UC Irvine, TR 187 (1982).

SQUIRES, S. L. (ED.):
Working Papers from the ACM SIGSOFT Rapid Prototyping Workshop.
SE Notes ACM SIGSOFT. 7(5), (1982).

STREICH, H., K.-H. SYLLA, AND H. ZÜLLIGHOVEN:
Anmerkungen zum Prototyping.
Paper presented at the GI'83 Annual Conference. Proc. will be published by Springer-Verlag, (1983).

TAVOLATO, P., AND K. VINCENA:
A Prototyping Methodology and Its Tools.
(In this volume).