# NATURAL

Manfred Mönckemeyer and Thorsten Spitta

Schering AG

Müllerstr. 170-178

D - 1000 Berlin/West

Germany

## Table of Contents



## Abstract

This paper describes how prototyping is made possible by using powerful tools in a software engineering environment based on a phase model. Prototyping is considered to be a useful technique WITHIN a detailed concept of software production rather than such a concept itself.

## 0 Introduction

Prototyping is one of several new SELLING-EXPRESSIONS used in the same way as OBJECT ORIENTATED SOFTWARE, NON-PROCEDURAL LANGUAGES etc. Those terms are sometimes used to give well-known concepts a certain veneer of learning. Prototyping seems to be such a concept. NATURAL is a rather new tool, but contains some very old concepts, now used in a data base environment (ADABAS).

NATURAL is said to be an INSTRUMENT OF PROTOTYPING AS A NON-PROCEDURAL HIGH LEVEL LANGUAGE OF THE 4TH GENERATION OF PROGRAMMING LANGUAGES, whatever all this will be.

We, that is Schering AG, Berlin/Bergkamen, West Germany, speak less sophisticatedly about NATURAL and prototyping. On the other hand we think, both

- prototyping as a technique
- NATURAL as a tool

are very useful in a large EDP-department using software engineering methods.

## 1 Definition of Prototyping

### 1.1 The Term 'Prototyping'

In the engineering sciences 'prototype' is a well-defined term meaning the first really working version of a new type of product. A prototype has all essential attributes of the final product, and it is produced in order to perform final tests and improvements before starting mass production.

Since software is no mass product, in the context of software development the term of prototyping must be used in a different manner.

The claims of prototyping are

- quick and simple basis of understanding between the user of software and the developer,
- stepwise introduction of large system, to get the user familiar with his new working environment,
- installation of improvement procedures after analysis of prototypes.

We define: Prototyping is the development of software by means of

- experimental user interfaces (= prototype programs)
- fixed requirements between end of specification and introduction in user departments
- as much tool-supported design and implementation as technically possible to shorten time between specification and implementation

in order to reach well accepted software.

In defining this, we know that

- the degree of acceptance can never be perfect because it depends not only on tools and organization but also on human behaviour.

- software has a very wide variety of complexity. Therefore prototyping is dif-

ferent for certain classes of problems.

TOOLS for prototyping must fullfil certain claims:

I. quick simulation of the end user interface, that is

   1. screen/list layouts on the really used medium
   2. transaction sequences on a real screen
   3. readable specifications of user-defined internal algorithms (calculations, plausibility checks of input data etc.)

II. quick and safe development (or handling) of the data base; this implies the use of a flexible data base management system with an integrated data dictionary.

III. quick realization of important parts of the system (strategy: most critical component first).
   See: [FREEMAN].

The AIMS of prototyping are:

- support of learning processes of
  - end users ==> about solutions with EDP
  - system analysts ==> about precise requirement specifications
- correct software, which is congruent with its specification.

## 1.2 Prototypes and Prototyping Activities

In the meaning of the claims above we define

- a PROTOTYPE is a system developed for experimental use only; it is a prototype program or sub-system as part of the final system

- prototyping ACTIVITIES (syn.: explorative prototyping; see [FLOYD]) to apply powerful tools during project development in order to reach

| quick and clean constructed software. |
|---|

Production of a prototype of the EDP-system to be developed may be effective, if the problems to be solved are very complex or if there is no experience in solving those problems. Also a prototype sub-system containing the essential features of the user interface may be used for teaching purposes before the final product is installed.

However, for most EDP-projects we do not consider the development of a prototype (sub-)system to be useful. In those projects we apply prototyping activities, which will be extensively described below.

By now the reader might have the impression that every usage of powerful design and implementation tools in software development is a prototyping activity. This is indeed intended.

We extend the range of prototyping activities from the requirement to the realization phase, because it is an old experience of SOCIAL sciences that user's requirements cannot be fixed for a longer time (see [CYERT]). So prototyping activities are part of an exploratory strategy intended to shorten time between specification and realization phase.

Our argument from the viewpoint of a large mainframe user is that the tool-landscape in this area for quick and cleanly constructed online software on an integrated data base, had been so poor until a few years ago, that now with

- new tools
- modified phase models

there is the possibility of reaching

- better acceptance
- quickly reconstructed improved releases

of software.

This we have defined as prototyping in 1.1.

## 2 Project Model with Prototyping

The model of prototyping contains

- the organization of software development,
- tools of development,
- quality inspection and improvement.

This model aims at

> as many prototyping activities in software development as possible.

## 2.1 Project Organization

We defined 6 phases of software development:

1 rough concept — detailed analysis of the organizational concept and first structuring of the functional and data base aspects of the application system

2 fine concept — requirement specification for each EDP-package; in this phase prototyping activities are performed depending on the complexity of the software to be developed

> phase 2 results in a contract with the end user

3 EDP-technique — modularization under EDP-aspects, data base design

4 realization — module specification and construction, programming, testing, integration

5 introduction — in the end user's environment

6 optimization — correction of faults detected in phase 5 and improvement of system performance.

In order to avoid endless loops between specification and realization, one rule must be strictly obeyed:

> Never change requirements after the end of phase 2!

This is supported by a project library. Without authorization by the firm's project management it is technically impossible to update documents of finished phases.

## 2.2 Software Development Tools for Prototyping

Schering uses general tools for software development. Prototyping is performed in connection with those tools.

The project organization is supported by a project library system distributed on a master-computer with a data dictionary and the slaves. The developer's interface is solely the slave. This distributed project library is only one of several possible solutions (for details see: [SPITTA]).

Since, especially during the first phases of development, the production of text documents is dominant, the slaves must provide a powerful text editor. Access to the host should be possible, too, in order to use master-tools (NATURAL is one of them).

To discuss the use of tools in the various phases, we distinguish between

- data producing software (DPS)
- query software (QS).

### 2.2.1 Requirement Specification and Design of Data Producing Software

DPS affects the firm's integrated data base by creating respective updating central data. During project development the data base grows if new object types are defined or new data elements are added to existing ones. Therefore in phases 1 (new types of objects) and 2 (new data elements to existing types of objects) a data model of the project is produced to see influences on the firm's data model. User views must be gathered with the data dictionary.

Because the process of finding new or updated data elements is not evident, the requirement specification contains a structured design of functions and data. The tool used for this is the data dictionary. Development and maintenance of the data model are supported by additional programs on the host, which check integrity of input-output data of functions and other relations between functions and data.

Prototyping activities are not primarily used to support quickness of phase 2 but to support the understanding between analyst and end user on how the system will work. Our experience has shown that this cannot be achieved when the end user interface is designed on paper.

Screens and listprints must be designed using prototypes of single programs and sequences of maps on the screen. This increases the probability that the requirement specification really expresses what the user needs.

The analyst must be able to implement and change the prototypes in a quick and easy manner. NATURAL fulfills this requirement.

Screen layout is supported by the NATURAL Online Mapping Utility. This utility allows the map to be edited by means of symbols placed where the fields are to appear on the screen (field attributes are defined by placing user-defined control characters). The map may then be tested and saved. Finally an object module may be generated automatically.

For the purpose of documentation the maps are copied into the requirement specification, and the logical flow is described using state diagrams (see: [DENERT]), which can be easily understood by both the end user and the analyst. Data and functions (processes) are specified in prose or pseudo-code, but strongly subdivided into functional trees (tools: see above).

As a result of the use of this powerful tool there are at the end of phase 2 executable object modules of the maps, which will be used during further development without any change. In this case prototyping directly produces parts of the final product.

Simple list programs can be written very easily, too. Since NATURAL was originally developed to be a query language, real data base accesses are quickly implemented. Thus there is the possibility of displaying maps and list-prints filled with data. This will additionally improve understanding by the end user.

**All prototypes of DPS programs are to be thrown away later.**

Thus there is no use in implementing complex algorithms. And of course, no data base updates may be done by the prototype!

The NATURAL Security System is used to define

- which commands and
- which data base accesses

each user is allowed to perform.

In phase 3 the data base will be designed on the basis of the data model. The common ADABAS utilities combined with DATA MANAGER PRODUCE-commands are used as tools.

### 2.2.2 Design of Query Software

QS does not have an impact on other applications by updating the firm's data base and is mostly simply structured.

Selected NATURAL query commands are accessible to the end user, who is thus enabled to do some simple queries by himself without involving the EDP-department. Therefore this type of QS is not subject of this paper.

For a DPS, phase 2 may last relatively long because of the necessary synchronization process with the data base administrator. This is not necessary for a QS. Only the user interface and the query requirements are specified in phase 2 using the same tools for prototyping as described in 2.2.1.

**Prototypes of QS programs will not in any case be thrown away.**

This depends on the complexity of the QS and will be decided in phase 3 after the module structure and the interfaces to the data base are specified.

### 2.2.3 Programming and Testing

In phase 4 modules are specified and constructed, coded and tested. Successfully tested modules are integrated into sub-systems and the sub-systems are tested. This process continues until the whole system is integrated and tested.

Productivity in this phase depends heavily on the correct use of software-engineering techniques and on the power of the tools used as well.

| activity | tools in general | tools (Schering) |
|---|---|---|
| specification | text editor with function keyboard, | PET/MAESTRO |
| construction | product standards, data dictionary, | NATURAL |
| module coding | code generation functions | COBOL ANALYZER & OPTIMIZER DATA MANAGER |
| module testing | standard stubs, testdata generator, compiler aided test documentation | |

Prototyping activities in phase 4 aim at quickness and transparency without giving up the certainty of developing correct programs.

When the sub-product PROGRAM CODE is to be generated, the project library system provides the programmer with a file on the slave already containing a program frame in the requested language. Coding the control structure of the program is aided by programmed function keys. E.g. if the repeat-function key is pressed a repeat-control-structure is generated:

COBOL:

```
REPEAT.
   _
   IF ??? GOTO ENDREPEAT.
   GO TO REPEAT.
ENDREPEAT.
```

NATURAL:

```
REPEAT
   _
   IF ??? THEN ESCAPE

CLOSE LOOP
```

'_' indicates the position of the cursor. The generated lines are protected against updating; the '???'-string may be replaced by a procedure.

As described above (see: 2.2.1), the object modules of all screen maps are already generated in phase 2. Also list generating statements exist in the prototype programs and may be copied into the module's source and used with only few modifications.

For the testing of modules, stubs are used to simulate interfaces to other modules.

Unfortunately our software engineering environment lacks any support for controlled and reproducible testing. Such a test support system does not exist in any industrial EDP environment we know.

## 2.3 Quality Inspection

Quality inspection is very closely combined with development tools, because it is impossible to examine programming and specification standards manually in a large EDP-department.

We improve quality by

- learning from the prototype
- learning from the user
- using tools as described in 2.2
- final phase inspections by a project management institution.

## 2.4 Summary of the Model

Prototyping activities are recapitulated in the following table:

| phase | prototyping activity | tool |
|---|---|---|
| 1 rough concept | — | — |
| 2 fine concept | simulation of transaction sequences, screens, list-prints, state diagrams | NATURAL (ADABAS), DATA MANAGER, PET/MAESTRO procedures |
| 3 EDP-technique | — | DATA MANAGER |
| 4 realization | quick and clean development, module and integration testing | PET/MAESTRO procedures, NATURAL, DATA MANAGER |
| 5 introduction | very complex systems only: stepwise introduction using prototype sub-systems | — |
| 6 optimization | — | — |

## 3 Experiences and Concept of NATURAL Usage

Our concept of NATURAL usage for

- prototype programs
- prototyping activities

was strongly influenced by first experiences with NATURAL.

Therefore both aspects of NATURAL

- first experiences
- usage concept in future

are reported in a mixed manner.

### 3.1 First Experiences with NATURAL

When Schering began to use NATURAL and started a pilot project in IV/1982, we tried to participate in the experiences of other NATURAL users by being consulted. We soon got the impression that most firms use NATURAL as an instrument of quick and dirty realization.

It is a severe error to believe that quality inspection is not necessary in a prototyping environment. (The expression RAPID PROTOTYPING has promoted this opinion.)

In addition to that, it became evident that the design of the language contains several inconsistencies and dangerous respective unpleasant features.

So we decided to TAME NATURAL by programming guidelines and quality inspection procedures. Also we had to develop a user profile concept for an extensive use of the NATURAL security system.

The pilot project now reached the end of phase 4, and it may be stated that the combination of

- Schering programming rules (tool supported by PET/MAESTRO procedures) and
- NATURAL Security System

is a very efficient aid for

- prototyping by quick development as well as
- disciplined production of programs to be easily maintained.

In the meantime the concept of integration of NATURAL into our software-engineering environment was developed as decribed above.

Though some parts of the tool support have not yet been completely realized,

- the dialogue with the end user and his understanding
- the quickness and transparency of program construction and coding

have improved a lot compared with projects developed formerly using COBOL, PL1, FMS, DMS and other non-integrated tools on an IMS data base.

### 3.2 Schering's Concept of NATURAL Usage

We would like to emphasize that there are many similar tools, which can also be used with the same development concept as required in 2, e.g. we are going to install a CICS online mapping utility on PET/MAESTRO.

Nevertheless, the topic of prototyping with NATURAL was not simply selected because Schering uses it, but because NATURAL is more than just a programming language. It provides program development tools as well as an environment, which - with the exception of the editor - allows quick and easy programming and testing, thus making NATURAL well-suited for prototyping in the sense of 1.

In phase 4 (realization) the NATURAL Programming Language is used in combination with COBOL subroutines, because of

- faster execution times
- some more usable features of COBOL, esp. table handling, which will be improved in later releases of NATURAL.

As far as training is considered, the combination of both programming languages has advantages because of their syntactical similarity.

### 3.3 Conclusion and Future Plans

We summarize the concept of NATURAL usage as a tool of prototyping:

| phase | usage | date |
|---|---|---|
| 2 | - simulation of end user interface | I/83 |
| | - DPS prototypes (throw-away-programs) | IV/83 |
| | - QS prototypes which are to be finished after a very short phase 3 | IV/83 |
| 4 | - programming language for | |
| | . screen/listprint interfaces | II/83 |
| | . queries | IV/83 |
| | . simple batch programs | II/83 |

NATURAL is going to be used under the control of Schering's project library.

In 1985 an end user concept will be realized to enable users to retrieve information from the ADABAS data base independently from the EDP-department.

## 4 Experiences with a Prototype System

During the development of the project library (see 2.2), which was one of the most complex projects ever realized at Schering, a prototype sub-system on the PET/MAESTRO-slaves was implemented.

The user interfaces and several essential functions were prototyped, and several other projects used that prototype. In this way we were able to gather experience as well as to teach this new technique ONLINE before the system was realized.

More than 60 system analysts, programmers and other future users were trained on the prototype. Several pilot projects applied the functions of the prototype. Practical experiences with the prototype were of great value for the final specification of requirements and the technical knowledge of how to implement the slave functions as well.

After half a year this prototype was completely replaced by the final system, which has been in use since I/83.

## A Literature


CYERT, R. M., AND J. G. MARCH:

A Behavioral Theory of the Firm.

N. J.: Englewood Cliffs 1963.

DENERT, E.:

Specification and Design of Dialogue Systems State Diagramms.

Proc. Int. Comp. Symp. Liege 1977.

Amsterdam: North-Holland 1977. 417-427 (1977).

FLOYD, CH.:

A Systematic Look at Prototyping.

(In this volume.)

FREEMAN, P.:

Design Fundamentals.

State of the Art Report, Structured Analysis and Design, INFOTEC. Vol.2, 117-131 (1977).

SPITTA, T.:

Eine verteilte Projektbibliothek in Verbindung mit einem Data Dictionary.

Software-Technik-Trends (GI-Fachgruppe Software Engineering). Heft 3-2, 60-82 (July 1983).