FORMALIZED SOFTWARE DEVELOPMENT IN AN INDUSTRIAL ENVIRONMENT

Otthein Herzog
IBM Germany, Dept. 3100
P.O.Box 80 0880
D-7000 STUTTGART, F. R. G.

ABSTRACT

In the IBM Boeblingen Laboratory some software was experimentally developed in the framework of a "traditional" life-cycle model where precise semantics were introduced very early in the development process through the use of a formal specification method.

As a typical example for these ongoing efforts, the development of a medium-size software product is presented where a first informal global data flow specification was described using simple graphical conventions. The result of this development step was refined and formalized using the formal specification technique proposed in [JON80].The specification and design language SLAN-4 [BEI83] was used to document this specification. The experiences are outlined which were gained by this development approach.

1. The Software Life Cycle - a Specification in itself

There is a widely accepted software life-cycle the steps of which may be called differently but which usually are established in the following way:

1. Requirements collection: evolving a functional concept.

2. Specification: describing an appropriate global system architecture.

3. High Level Design: structure the system.

4. Low Level Design: getting the algorithms straight.

5. Implementation: coding.

6. Test: probing functions and quality.

7. Maintenance: maintaining proper functioning.

Obviously, this is a specification of a software development process which can be implemented in many different ways - and there are many methods and tools stating exactly this fact. However there does not seem to be a coherent method yet which has been widely accepted since different methods are normally used in the different development phases, and in addition, most of the methods appear to be aimed at specialized development areas such as certain types of applications or of system oriented software, e. g. teleprocessing protocols.

Before proceeding to the approach taken in the experiment some observations about implementations of this development process in general are worthwhile to be mentioned:

- **Documents**

  The outcome of each development step should be a document or even a collection of documents which serves external or internal purposes, e. g.

  - a functional description of a proposed software system to be used in negotiations with future users,
  - a functional system decomposition to be used by the designers,
  - a system specification to be used by designers as input to base the high level design on, by test designers to plan the test phases or by technical writers to develop the documentation,
  - the programs themselves.

  The collection of the documents written during development constitutes what is called software.

- **Formalism**

  It is commonly accepted that the very first concept of a software system is conveniently described in an informal way. But in general the managers, designers, analysts and programmers got used to work with documents written in natural language, mostly English, which is in addition a foreign language for many people in the programming community.

  As a common practice, these informal documents are augmented by flow charts, data flow diagrams, data definitions, and programming language control constructs (pseudocode) to describe additional details.

  In this way there is an increasing formalism in the documents resulting from the development steps, but a complete formal description can be accomplished only at the coding level. There are some consequences of this approach: the documents

  - tend to be very long,
  - are ambiguous, inconsistent and incomplete,
  - cannot be refined without implicit assumptions and interpretations.

- **Validation and Verification**

  There is no question that each subsequent development step has to be an implementation of the preceding phase. But given the restriction that the early development steps are not documented in a formal way it is not possible to effectively use automated verification in these phases. Inspections and walkthrus are used instead facing all the limitations of the human mind confronted with large volumes of documentation.

  This basically leads to the high specification and design error rate of up to 60% [MEN82] which is uncovered during the regular use of software. Although up to 50% of the development costs are usually spent in the test phase these specification and design errors are detected very late in the process and thus are very costly to correct since it might even be necessary to back up several development steps to effectively correct this type of errors.

Taking all these observations into consideration the formalization of the early development steps should be at least a partial solution to this problem. It should improve the early error detection and should help to avoid early errors.
Which are the properties of methods to satisfy these strong requirements? They

- enforce a coherent, precise and minimal description of a system's behaviour which is complete in respect to its purpose,
- are interface oriented,
- describe the essentials of the problem solution, not implementation details,
- offer non-procedural description elements which usually lead to conciser system descriptions than algorithmic ones,
- stress the description of system interdependencies,
- build a bridge of understanding between the users, requirement planners, designers, and implementors.

The following objectives refine some of these properties: "Good" specification and design methods

- add discipline to the specification and design process,
- separate "WHAT" from "HOW",
- rigorize the system interface definitions,
- encourage levels of abstraction,
- promote hierarchical architectures,
- support maintainability of documents,
- are independent of hardware.

In addition it is necessary to support such a method by a language which can be implemented in an environment providing the necessary tool support. The items mentioned in [MAR83] give an impression of the important issues:

- The specification language must be rigorously defined.
- There has to be an interactive graphics facility.
- A mathematical basis allows for checking the logic structure.
- Rigorous refinement steps must be possible.
- Ultimately, code must be generated from the specification.
- The definition levels must be integrated: One language has to cover all development phases to ease the maintenance of front-end documents.
- Integrated top-down and bottom-up specification must be possible.
- An evolving library of reusable parts must be available.
- Integrated checking must guarantee interface consistency.
- Documents are easy to change, also by persons which did not create them.
- All elements of a system should be traceable.

2. An experiment to formalize early development phases

In the IBM Boeblingen laboratory the specification and design language SLAN-4 has been developed [BEI83] which satisfies almost all the requirements against such a language. It provides constructs for the algebraic specification and implements also essential parts of the Vienna Definition Method ("VDM")[BJO78]. Some experiments were conducted to use this method and this language to determine the effects of the formalization of early development phases.

A medium-size software product including an interactive user interface was specified using a combination of a simple graphical notation for the global data flow and SLAN-4 for the formal specification of the operations to be performed by the individual modules. This formal specification was expressed using SLAN-4 syntax with **pre-** and **post-**conditions for each of the operations.

The Software System

The system addresses a networking environment, resides on a host machine and allows a network administrator to

- maintain a data base on network resources (repository of the hardware and software configuration),
- report on the stored information,
- retrieve and send data from and to nodes,
- get status information from the nodes, and
- send messages to the nodes.

It offers the user an interactive front-end with guidance to the main system functions through a small number of selection menus.

The total size of the system is 17,300 lines of an IBM internal very high-level language code, not including 4,000 lines of HELP text, which is available on-line during a session.

Specification and Design

The development was done top-down with very few iterations and included a mixed design strategy:

1. The global flow was designed using an outside-in method starting from the functional description of the user interface.

2. The functional specification resulted in a hierarchical decomposition of the system including the panels required for the user interaction within the individual functions. Again, the user interface determined here essential parts of the specification.

   For the functional specification itself, the global data state and the data interfaces for each module were specified. Then the elaboration of the pre- and post-conditions allowed the developers to check the completeness and consistency of the interfaces.

The specification phase was completed after a thorough inspection of the specification document which was very effective because of the unambiguous description of the system.

3. For the low level design step, the data object declarations were carried over to the design/implementation language and the algorithms were developed using well-known sequencing control structures to implement the specified functions.

   The final layout of the selection menus and the data entry panels was defined in such a way that a prototype of the system was the outcome of the low level design. This step was also concluded by a design inspection.

Implementation and Test

1. A very high level implementation language was used to directly implement the design. This language offered very high level data types, such as sets, bags, lists and arrays.

   The development team was thus able to also use the data types selected during the specification phase in the low level design and also in the implementation phase.

2. After the code inspections the test of the system was carried out in different steps:

   a. Each developer performed a structural unit test taking the internal structure of the code into account.
   b. Then the code was transferred to a test team which subsequently did a function test against the specification. The next test phase was a component test against the user documentation to assure the proper cooperation among the different functional components. Finally a system test was performed varying hardware and software configurations.

   The test period lasted four months. The errors found were formally reported and the valid errors were corrected in all affected documents including the specification, since this document was supposed to be the description of the system for the maintenance team.

Evaluation of the experiment

The following points resulted from the methods and the languages used:

- The method was applicable to a software system without heavy external interfaces.
- Module interfaces could be explicitly documented.
- Abstract data types were successfully used.
- Each module could be described in a non-procedural way independently from the final implementation.
- The precise module semantics and the complete interface description led to very stable interfaces between the individual modules which were in most cases only changed by new requirements.
- During four months of function, component and system testing, 5.84 valid program errors were found per 1000 lines of code. This compares to 20 to 40 errors usually found during such a test cycle for software of comparable complexity. The main reasons for this result were the early formalization which helped to avoid errors and to reduce the error propagation through the subsequent development steps and the availabilty of the same high level data types in the specification, design and implementation languages. This avoided errors introduced by implementations of abstract data types.
  However, an analysis of the errors found showed the interesting result that an integrated formalized description of the global data flow would have avoided some errors which were found during the test cycle.
- There was also a considerable positive effect on productivity.
- Although the method and the language was new to the majority of the development team, the early development phases did not need more time than "conventional" specification and design methods.
- The specification was invariant against small requirement changes. During the development a considerable change in the requirements for the user interface could be easily incorporated in the existing specification document.

3. References

[BEI83] F. Beichter at al.: SLAN-4: A Language for the Specification and the Design of Large Software Systems.-
IBM Journal of Res. and Dev. Vol 27(6), Nov. 1983, p. 558 - 576

[BEI84] F. Beichter at al.: SLAN-4: A Software Specification and Design Language.-
IEEE Trans. Software Eng. Vol. SE-10(2), March 1984, p. 155 - 162

[BJO78] D. Bjorner, C. B. Jones (Eds.): The Vienna Development Method: The Meta-Language.-
Springer (1978)

[COT84] I. D. Cottam: The Rigorous Development of a System Version Control Program.-
IEEE Trans. Software Eng. Vol. SE-10(2), March 1984, p. 143 - 154

[JON80] C. B. JONES: Software Development - A Rigorous Approach.-
Prentice Hall (1980)

[MAR83] J. Martin: Fourth Generation Languages, Vol. 1.-
Savant Research Studies (1983)

[MEN82] K. S. Mendis: Quantifying Software Quality.- Quality Progress, pp. 18-22, May 1982.

[PEP84] P. Pepper (Ed.): Program Transformation and Programming Environments.
Report on a Workshop directed by F. L. Bauer und H. Remus.-
Springer (1984)