Prototyping of Software-Architectures - A Seamless Transition into the Final System

Horst Lichter
University of Stuttgart
Department of Computer Science
Breitwiesenstr. 20/22
D-7000 Stuttgart 80

Abstract

This paper introduces a prototyping approach which considers the architecture of programs as the central aspect modelled by a prototype. This prototypes are called architecture prototypes. An architecture prototype can be used to develop an appropriate architecture to a given problem very early in the design process. A special development process model for architecture prototyping is introduced which allows a seamless transition from the prototype into the final system.

1. Introduction and Motivation

Prototyping is by no means a new concept. In many engineering disciplines, e.g. mechanical or civil engineering, *prototyping* is a technique that is invariably employed to improve the calculability of new projects that involve risks (cf. Lantz, 1986). For similar reasons, prototyping has also been adopted as a technique in software engineering. An essential requirement on prototypes is that they should be available very early in the development process. This leads very often to a *quick-and-dirty* development of prototypes. But only *presentation prototypes* which are planned as throw-away-prototypes must be constructed in this fashon. The quality of a developed prototype should not be neglected during the design process, especially if the prototype should be enhanced evolutionary into the final system.

Experiences as published in Kieback et al. (1992) or Gordon (1991) show that even if a prototype is to be developed as a throw-away-system, the expectations of clients often enforce the evolution of such a quick-and-dirty-prototype into the final system. The adoption of quick-and-dirty-prototypes as technical building blocks in the further course of the development process is something of a problem, if not disastrous for the project as a whole. The very nature of these prototypes makes them unsuitable as a sound basis for producing a high-quality architecture appropriate to the problem in hand. The terminology and the concepts of prototyping are explained in detail e.g. in Floyd (1984) or Budde et al. (1991).

1.1 Overview of this Paper

This paper describes the results of an ongoing research project which considers the *architecture* of programs as the central aspect modelled by a prototype. This kind of prototypes are called *architecture prototypes*.
In the second part architecture prototyping is introduced and some objectives are formulated. The third part describes a development process model which is based on architecture

Proceedings of NordData '92 European Conference on Information Technology,Tampere, Finland, 1992.

prototyping. Part four explains some basic ideas on architecture modelling. Part five contains a short description of a concrete application of this approach.

2. Architecture Prototyping

The result of the design activity is the architecture of the program which is defined as follows:

> A *program-architecture* is the high-level structure of a program. It defines the static structure of the program, i.e. its components, the purpose of each component and the relations between them.The term *architecture* is used as a synonym.

Nagl (1990) uses the term “discrete architecture paradigm” to express that all components of an architecture can be identified without a closer look into their internal construction. This is a very idealistic view on the design process which strongy seperates *system-design* from *detailed-design*. This does not agree with the everyday experience. The architecture of a program is developed during the process of intertwining both activities, *system-* and *detailed-design*. The design process naturally starts with the system-design. After the basic components have been identified the datailed-design of these components may be done. This can create or effect other components of the architecture. In this way the architecture is stepwise developed.

Prototyping is a method to get information about a critical and unclear aspect by constructing a model of this aspect which is used and evaluated. This is the motivation for architecture prototyping too. The design of the program architecture is a very riskfull activity in every software project because the quality of the architecture influences mainly the quality of the resulting program. Consequently it makes sense to build a prototype which concentrates on the architecture of a software system. The terms prototype and architecture prototyping are defined as follows:

> A *prototype* is an executable model of a special aspect of the final program.
>
> The construction of a prototype which models the program architecture, and the cycles of execution, evaluation, and modification of the prototype are called *architecture-prototyping*.

The architecture prototype is like any other prototype executable. In order to execute the prototype the components of the architecture must be prototypically implemented. After the evolution the architecture prototype defines the architecture of the *final program*.

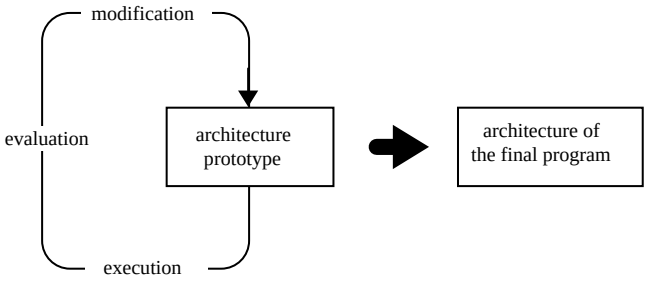

Fig. 1: The basic process of architecture prototyping

Architecture prototyping takes into account that both design activities, system- and detailed-design, are performed in cycles. The components of an architecture are consequently not only

identified but also implemented in a special *prototyping language*. In this way the designer has the possibility to discover how a component is embedded in the overall architecture. Every component is implemented in its prototype version up to a degree which permits to identify the relations to other components. The components are however not implemented in their complete functionality and with the required robustness. The prototype implementation of a component serves only to get information about the architecture which cannot be obtained otherwise. An architecture prototype enables the designer to analyze important design qualities, e.g. maintainability, conformity to the problem in hand or reusability, which are essential for every software system.

3. PDSC: A Process Model for Architecture Prototyping

PDSC (**P**rogram **D**evelopment by **S**tepwise **C**ompletion) is a development process model which is based on architecture prototyping and is introduced in Ludewig et. al. (1987). It is characterized as follows:

The program architecture is designed based on an informal requirements specification. The resulting architecture description serves as the starting point for the construction of the architecture prototype. The architecture prototype is executed, evaluated, and modified. If there are components in the prototype which are stable over some cycles of evaluation and modification (the interface of this components doesn´t change any more), these components can be implemented in the programming language of the final system *(final lanquage*). After the final implementations have been tested they are integrated in the architecture prototype.

3.1 The Process of Stepwise Completion

During the process of exchanging prototype implementations through final implementations it is necessary to ensure that the architecture prototype still remains executable. At the beginning of this process the architecture prototype consists of components which predominantly are implemented in the prototyping language. Only components which are reused or which are easy to implement in the final language are not used in its prototype implementation.

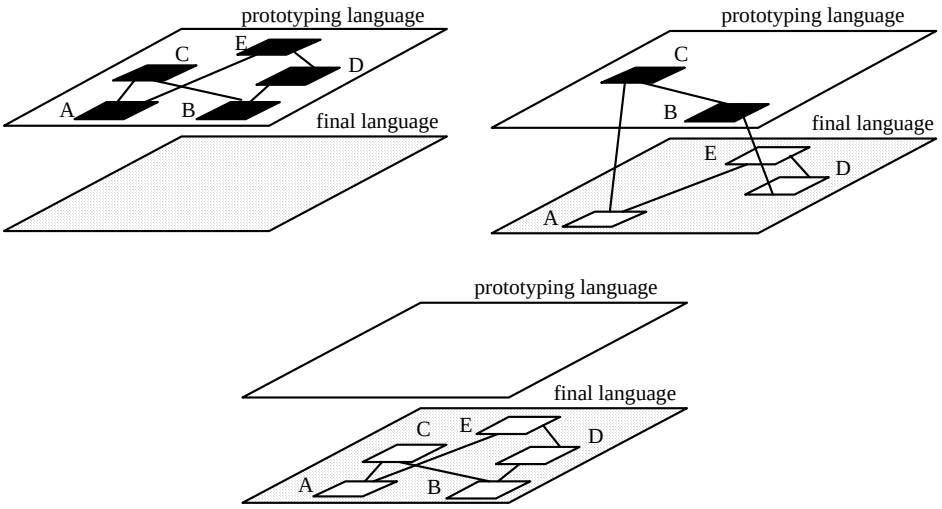

Fig. 2(a/b/c): Stepwise completion of an architecture prototype

In the course of the development the prototype implementations of the components are stepwise replaced by the final implementations. Only the structure of the interface, i.e. the relations to other components and the signatures of offered operations, can be transformed from the prototype version into the final implementation of each component. The prototype implementations are not used and thrown away. Figure 2 illustrates this process.

Every component in the presented simple example is, at the beginning of the development, implemented in the prototype language (Fig. 2/a). During the development the already stable components (in the example the components A, E, and D) are implemented in the final language corresponding to their prototype description (Fig. 2/b). After their integration into the architecture prototype they build together with the components B and C which are still in their prototype description a new version of the prototype. The final implementation of the components A, E, and D are integrated in the architecture prototype according to their prototype descriptions. The final program is completely implemented if none of its components are in the prototype description (Fig. 2/c).

3.2 Activities in PDSC

As every development process model PDSC defines special activities which has to be done during the development. The following figure shows the activities and the produced documents.

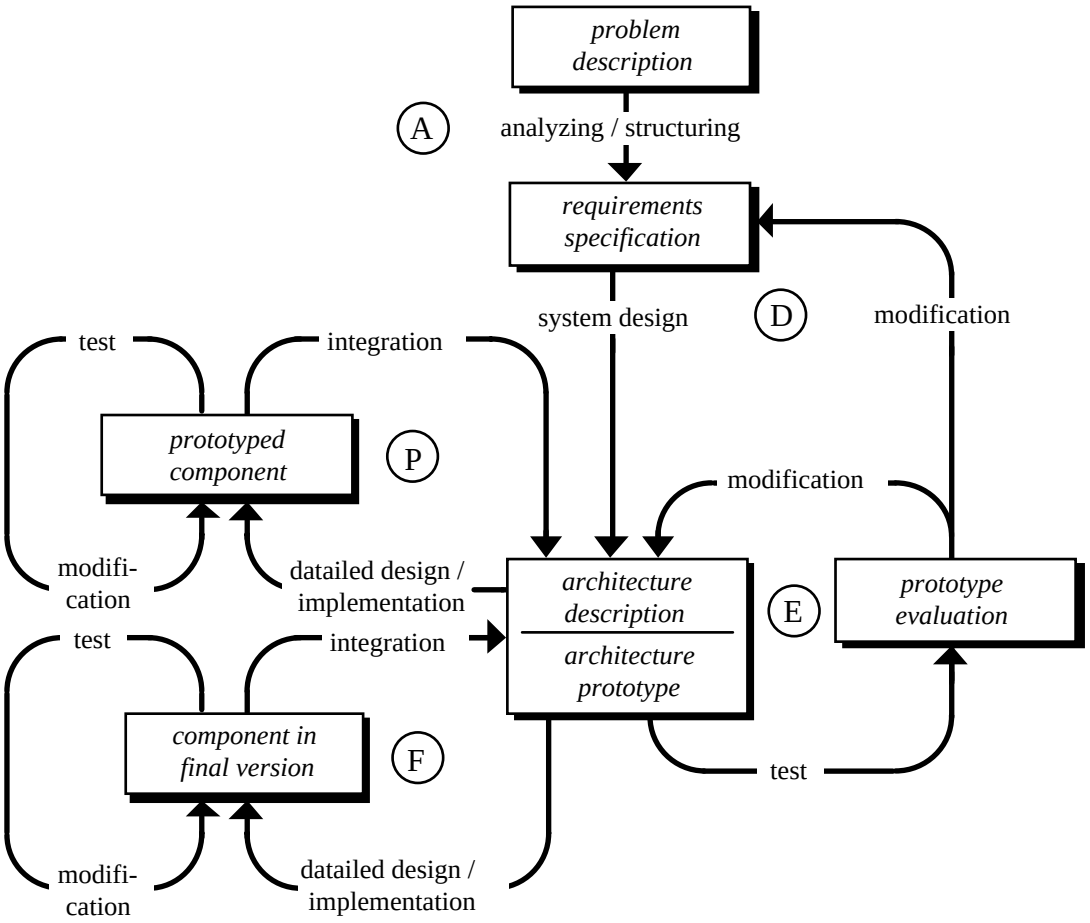

Fig. 3: The activities in PDSC

The activities can be described as follows:

A Analyzing of the problem in hand and structuring of the requirements. The result is an informal requirements specification.

D Design and modification of the software architecture.

P Implementation of the components in the prototyping language, test, and integration in the architecture prototype.

E Evaluation and modification of the architecture prototype.

F Implementation of the components in the final language, test, and integration in the architecture prototype.

The process starts with the activities A, D, P, and E. Afterwards the order is arbitrary.

4. Architecture Modelling

In order to construct an architecture prototype which is a model of the architecture it is necessary to find out what are the relevant aspects of an architecture to be modelled. A closer look shows that there are three levels of descriptions concerning architectures. They are called *scheme-level*, *model-level,* and *architecture-level*.

Scheme-Level: The types of components and relationships which may be used to develop models of architectures are defined on this level. For this purpose it is necessary to decide which types of components and relationships are relevant and which are not relevant for the model. The choosen types of components and relationships form the *architecture scheme*. It should be formulated in an adequate notation. Since the architecture scheme defines types of components and relationships an ER-notation may be used.

Model-Level: The models of architectures are constructed on this level. They have to agree with the architecture scheme. An architecture model is formulated in a language which permits to express all the possibilities of the architecture scheme. This language is called *architecture description language*.

Architecture-Level: The concrete architectures are located on this level. They are formulated in a programming language and must agree with their models. A concrete architecture defines however components and relations which have not been relevant for the model.

The following figure illustrates the relations between these levels by way of a simple example.

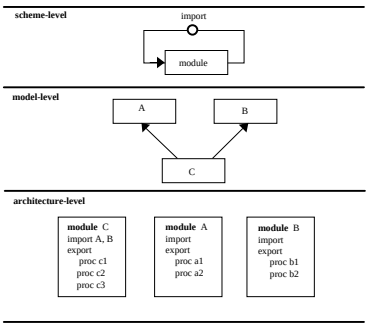

Fig. 4: Levels of description

The architecture scheme only offers the component type “module” and the relationship type “import” between modules. An architecture model consisting of the modules A, B, and C and the relations C imports A and C imports B is constructed according to this architecture scheme. The concrete architecture which is build according to its model futhermore defines all procedures which are exported by the modules.

5. ProSA: A System for Architecture Prototyping

ProSA (**Pro**totyping of object-oriented **S**oftware **A**rchitectures) is a special prototyping system based on PDSC and designed to construct prototypes of object-oriented architectures. ProSA can be described through a *system-triangle.* Introduced in Ludewig (1985), a system triangle illustrates the relations between concepts, methods, languages, and tools of a system. Figure 5 shows the ProSA system triangle.

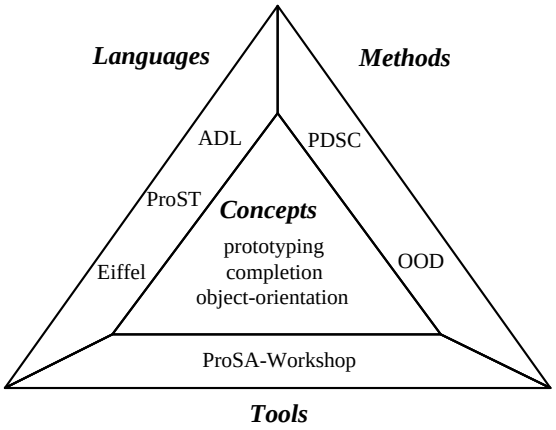

Fig. 5: The ProSA system triangle

The basic concepts are architecture prototyping, stepwise completion of the prototype, and object-orientation. The central methods are PDSC and object-oriented design (OOD). Three languages are used to construct an architecture prototype: an architecture description language (ADL), a prototyping language (ProST) and a final language (Eiffel). A set of tools called ProSA-Workshop supports the development process.

As described in section four an architecture scheme defines types of components and relationships which may be used to build architecture models. In the rest of this section an architecture scheme for object-oriented program architectures is explained, the ProSA languages are introduced, and the ProSA-Workshop is described.

5.1 A Scheme for Object-Oriented Architectures

The architecture scheme is based on the concepts of typed object-oriented languages. It offers the component type *cluster* which can be used to group *classes*. A class defines *features* which are private or may be *exported* by the class. Features are specialized in *routines* and *attributes*. Routines themselves are specialized in *functions* and *procedures* which may be *concrete* or *virtuell*.

Classes may be structured with the *inheritance* relation which produces specialized and generalized classes. A class may *redefine* features which it inherits from its superclass.

Virtuell classes define general features for all its subclasses. The implementation of these general features is deferred and must be done in concrete subclasses. Virtuell classes can be viewed as specializations of classes. A further specialization of the component type class is the *generic class*. Generic classes have formal generic parameters which makes them flexible and reusable. A class which is generic and virtuell is called a *virtuell generic class*. A detailed description of object-oriented architectures may be found in Meyer (1988). The following figure shows the graphical representation of the architecture scheme.

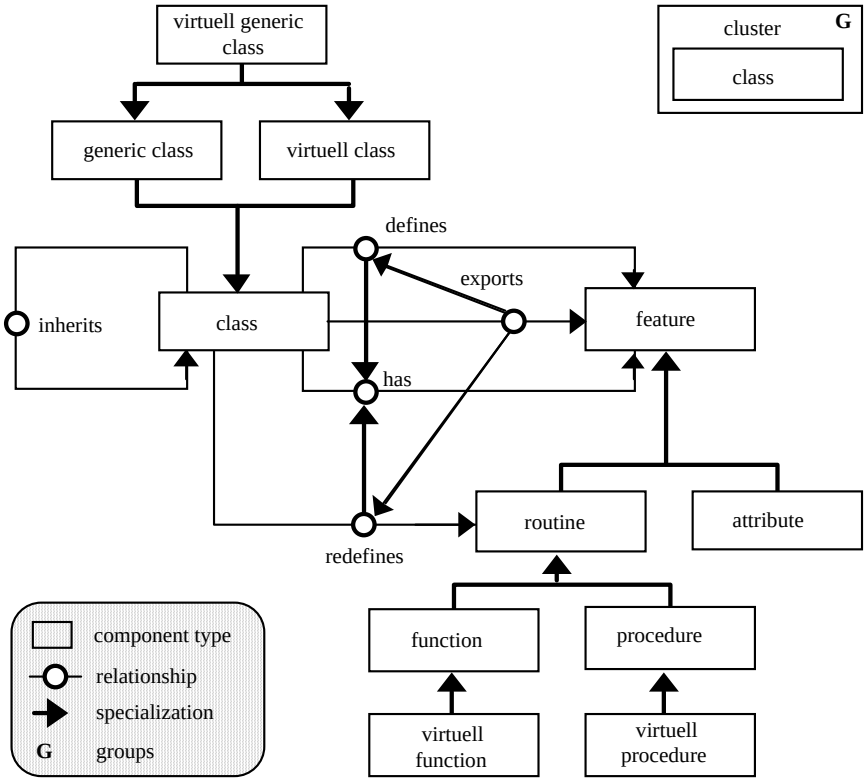

Fig. 6 : Scheme for object-oriented architectures

5.2 The ProSA-Languages

As mentioned before three kinds of languages are necessary for the activities in PDSC: an architecture description language, a prototyping language, and a final language.

Every final language should satisfy requirements which improves the quality of the produced programs like strong typing, seperate compilation, fast code, and readable syntax. I have choosen the language *Eiffel* (cf. Meyer, 1992) among the typed object-oriented languages because it satisfies these requirements. Furthermore it offers a lot of other important concepts, e.g. assertions on classes and routines or exception handling.

Architecture models based on the presented scheme for object-oriented architectures are formulated in the architecture description language ADL. It is a graphical language and a tool called architecture editor is used to edit architecture descriptions in ADL.

Standish (1982) defines a list of requirements which have to be satisfied by a prototyping language. Important requirements are:

- The language has to be interpretable. This avoids the edit-compile-run cycles.

- Incomplete programs must be executed.
- It should not be necessary to declare used identifiers, i.e. the language should not be type checked.

Furthermore a prototyping language for ProSA has to be object-oriented since object-oriented program components should be prototyped. Smalltalk-80 is an object-oriented language which satifies the listed requirements and has proved to be excellent for prototyping (cf. Diederich, 1987). Since the transition from the prototype version of a component into its final version which is implemented in Eiffel should be as easy as possible I have extented Smalltalk-80. The extention is called ProST and has some new features, e.g. generic classes, declaration of parameter types for methods and instance variables, which make the transition to the final language Eiffel very easy.

5.3 The ProSA-Workshop

The activities in PDSC have to be supported by special tools. Since every program component is formulated in three different descriptions (architecture, prototype, and final description) and every description is developed in versions and possibly in variants the ProSA-Workshop has to support the configuration managment for the architecture prototype which is typically executed in its completest configuration, i.e. every component is used in the latest version of the completest description.

The configuration management can be supported by a database management system which allows to administrate versions, variants, and configurations. These database management systems are called software-engineering database management system (cf. Glinz, 1985). In the ProSA-Workshop the object-oriented database mamagement system GemStone is used as the central component of the workshop. Every piece of information is stored in the database. Figure 7 shows the structure of the ProSA-Workshop.

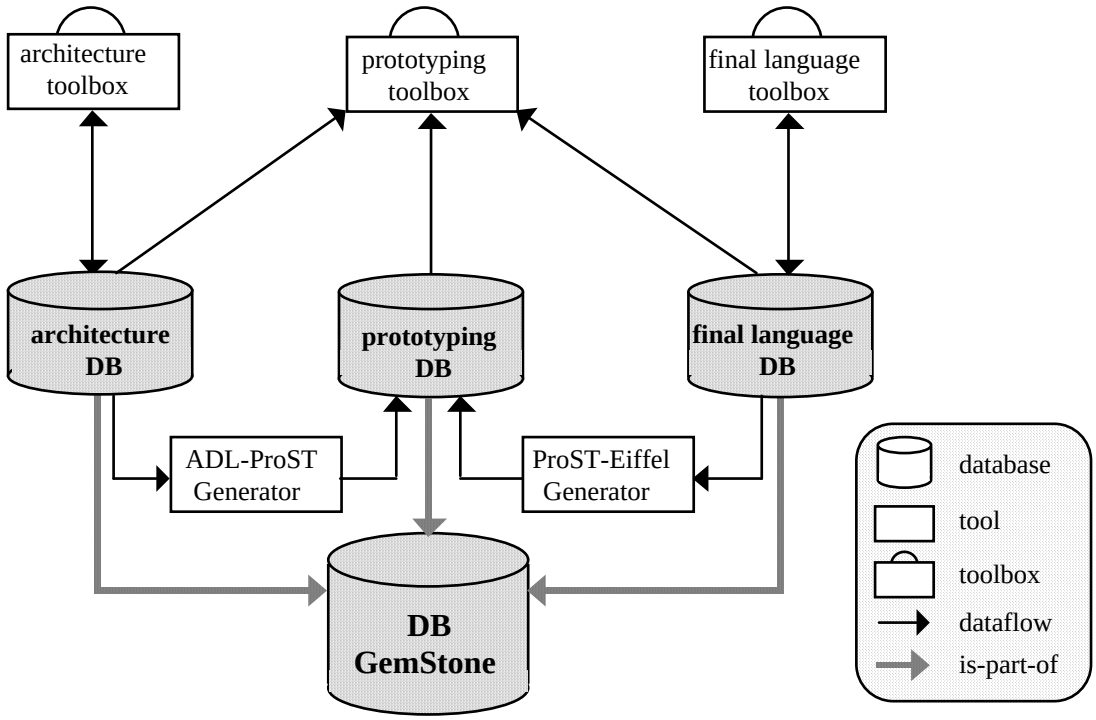

Fig. 7: Structure of the ProSA-Workshop

In the ProSA-Workshop the tools are grouped in four toolboxes:

The architecture toolbox: It contains tools for editing and analyzing architectures formulated in ADL, e.g. a graphical architecture editor.

The prototyping toolbox: It contains both editor and interpreter for ProST and the prototype configurator.

The final language toolbox: It contains editor, compiler, and debugger for Eiffel.

The transformation toolbox: It contains a generator for the transition from ADL to ProST and a generator for the transition from ProST to Eiffel.

6. What has been Done and What remains to Do

Up to now the concepts and methods as described in this paper are developed and formulated. An initial set of tools for the ProSA-Workshop has been prototypically implemented in Smalltalk-80 and C. This set of tools includes a graphical interactive architecture editor, an editor and compiler for ProST, the ProST-Eiffel generator, and a tool which produces the interface code needed to execute an architecture prototype consisting of ProST and Eiffel components.

At the moment an interface which links these tools to the GemStone database and the prototype configurator will be implemented. After this work has been done experiences with this prototyping approach have to be gained, analyzed, and evaluated.

References

Budde, R., Kautz, K. Kuhlenkamp, K., Züllighoven, H. (1991): **Prototyping - an Approach to Evolutionary System Development**, Springer-Verlag.

Diederich, L.G., Milton, J. (1987): Experimental Prototyping in Smalltalk-80. **IEEE Software**, May 1987, pp 50-64.

Floyd, Ch. (1984): A Systematic Look at Prototyping**.** In: Budde et al. (eds) **Approaches to Prototyping**, Springer-Verlag, pp 105-122.

Glinz M., Huser H.J., Ludewig J. (1985): SEED - A database system for software engineering environments**.** In Blaser, Pistor (eds): **Datenbanksysteme für Büro, Technik und Wissenschaft**, Informatik-FB 94, Springer, pp.121-126.

Gordon, S., Bieman, J. (1991): Rapid Prototyping and Software Quality: Lessons From Industry. **Technical Report CS-91-113**, Department of Computer Science, Colorado State University.

Kieback A., Lichter H., Schneider-Hufschmidt M., Züllighoven H. (1992): Assessing Industrial Prototyping Projects. **Proceedings of the HICSS-25**, Koloa Hawaii.

Lantz, K. (1986): **The Prototyping Methodology**. Prentice-Hall.

Ludewig J., Färberböck H., Lichter H., Matheis H., Wallmüller E. (1987): Software-Entwicklung durch schrittweise Komplettierung. In Requirements Engineering´87, **GMD-Studien Nr 121**, GMD mbH, Sankt Augustin, 1987, pp 113-124.

Ludewig, J., Glinz, M., Huser, H., Matheis, G., Matheis, H., Schmidt, M.F. (1985): Spades-A Specification and Design System and its Graphical Interface. In **Proceedings of the 8th. Int. Conference on Software Engineering**, London.

Meyer, B. (1988): **Object-Oriented Software Construction**. Prentice Hall, Series in Computer Science.

Meyer, B. (1992): **Eiffel - The Language**. Prentice Hall, Series in Computer Science.

Nagl, M. (1990): **Softwaretechnik**: Methodisches Programmieren im Großen. Springer Verlag, Berlin.

Standish, T.A, Taylor, T. (1982): Initial thoughts on rapid prototyping techniques. **ACM Software Engineering Notes**, Vol. 7, No. 5, pp. 160-166.