A methodology integrating formal and informal software development*

Barbara Paech
Institut für Informatik, Technische Universität München
Arcisstr.21, D-80290 München

Abstract

This paper presents a methodology integrating formal and informal software development. By distinguishing several *dimensions* and *perspectives* we structure the development process into stages and documents with different emphasis on formality. Formality is only be required for documents which can be related through formal techniques to the system design.

1 Introduction

Recently the applicability of formal methods in industrial software development has found significant interest. According to the different facets of a software development method (distinguished e.g. in [Hus94]), namely notation, techniques and methodology, three main approaches exist:

- formal notations: While formal methods define syntax and mathematical semantics for their notations, industrial methods mostly do not give a mathematical semantics to their notations, sometimes not even a complete syntax is provided. Thus several formalizations of informal notations have been given, e.g. for entity-relationship or data-flow diagrams [Nic94, Het93, PWM93].
- formal techniques: Formal notations are indispensable for formal methods. Based on formal notations also the techniques establishing the relationships between several documents can be formalized. One prominent example for such a formal relationship is *refinement*, others have been identified e.g. in the KORSO-project [PW94].
- formal methodology: The methodology describes how to use the notations and techniques in the development process. In the attempt to support the

*This work has been supported by the SysLab-project, conducted at the Technical University Munich, Department of Computer Science, Prof. Dr. M. Broy

development process with CASE-tools *process models* have been defined which identify the steps a development process is based on. These steps operate on a common repository of documents. By giving an operational semantics to these steps formality is achieved. For the process of requirements engineering such a process model has been given e.g. by the NATURE-project [JPRS94].

While process models are very useful for method comparisons and CASE-support, they do not address the question of how to integrate formal notations and techniques into the development process. The simplest solution is, of course, to apply the methodology of a well-known method (e.g. SSADM [DCC92]) on the formalized documents. This way, however, formality only improves single documents, and the process does not make use of the advantages gained by the formality, namely precise semantics and machine support. Also, since different documents serve different purposes during the process, it is not adequate to formalize all documents.
In the following we propose a more subtle solution to the above question. We have chosen a specific set of documents and steps which in our view is especially suited to combine formal and informal notations and techniques. Our choice has been inspired by the detailed methodology of SSADM and their more general counterpart described in [OHM+88]. It serves as an outset of the SysLab-project which aims at a scientific foundation of software development.

2 Dimensions and Perspectives in the Development Process

Our methodology covers requirements engineering and logical system design. It is particularly suited for the development of information systems, but may also be useful in other areas. In [Poh93] three *dimensions* of the requirements engineering process have beed identified :

- specification (opaque to complete),
- agreement (personal to common) and
- representation (informal to formal).

The specification dimension deals with the degree of requirements understanding, the agreement dimension captures the degree of acceptance of the specification by the different people involved in the development process and the representation dimension classifies the notations used to denote the specification. The process should lead from an opaque, personal and informal specifcation to a complete, commonly accepted and formal one.

We also add different *perspectives* to the specification and agreement dimension. We distinguish

- a data-oriented,
- a behaviour and process oriented and
- an objective oriented perspective

on the specification. The first two are also distinguished in [OHM+88]. The behaviour and process oriented perspective deals with the events the system reacts upon and the activities constituting the reaction. The distinction between data and behaviour is fundamental in all system models and also useful here: very often it is quite easy to give a complete specification of data, while the behaviour specification remains opaque for some time. In our view it is also important to capture the objectives underlying the different parts of the specifications. They give context information which is essential e.g. for a meaningful revision of the specification. Typically this kind of information is represented by informal text, but also formal relationships like the aggregation of activities into superactivities provide context information to specification parts.
For the agreement dimension the perspectives of the three typical participants of the development process should be distinguished:

- the business representative,
- the system user and
- the system designer.

The business representative is concerned with the embedding of the software system into the overall business processes, the system user is interested in an optimal integration of manual and automatic activities at his or her workplace and the system designer will implement the identified requirements by software. All of them have their own objectives and their own view on the data and the behaviour of the system. All of them use different representations for the specification of data, behaviour and objectives with different degrees of formality (see e.g. figure 1).

Perspectives	data	behaviour	objectives
business	results	business processes	business options
user	forms	tasks	work place options
system	data structures	transactions	design options

Figure 1: the matrix of perspectives

Although being desirable, it seems too expensive to record the full matrix of perspectives during the whole design process. At different stages some perspectives will be more important than others. This may very well depend on the project. In the following we describe one possible selection which seems reasonable in most information system development projects.

3 The Methodology

Since only the functional aspects of a system can be formalized, this section concentrates on the development of the system functionality. Similar to SSADM we distinguish *analysis* and *definition* in the requirements engineering process. Thus our methodology is divided into the three stages:

- requirements analysis,
- requirements definition and
- logical system design.

Requirement analysis[1] is mostly concerned with the business perspective. A model of (part of) the business is produced in order to understand the environment the software system is to be placed in. Requirement definition focuses on the user perspective. The functional requirements are captured by so called *system functions*, where system functions are the smallest processing units of the system to be called from outside. In the logical system design (emphasizing the system designer view) the system functions are associated to system components and the data and behaviour of the components is specified.
Although the stages to some extent correspond to phases of the development process, the main emphasis is on the products of the different stages. Underlying all stages are two documents

- a textual description of all project relevant concepts, the *glossary*, and
- a textual description of the requirements, the requirements catalogue.

Hypertext links should be used to establish the relationship between textual parts and their counterparts in the more formal documents. The glossary can also be organized as a meta-model as in [FL94].
As in STEPS [FRS89] our methodology embodies an *evolutionary* approach developing the system in versions. We call a consistent set of products of the three stages together with the glossary and the requirements catalogue a *version* of the system[2].

[1] also called business analysis in [OHM+88]
[2] or, more precisely, a version of the system model

3.1 Formality

Based on [BDD+92] we provide a mathematical system model covering distributed processes and explicit time. Similarly to the glossary, which gives an informal meaning to all conceptions encountered in the development process, the system model gives a formal semantics to all concepts of the documents. The formal semantics can be exploited in the definition of powerful CASE-tools. For example, at each stage the *consistency* of the documents can be checked. This is especially important, since the documents emphasize different perspectives. The formal semantics can also be used to verify the development steps of the logical design out of the requirement definition. However, to allow for a true integration of the perspectives of the different participants there always has to be a way to represent the formal documents informally.

3.2 Requirements Analysis

Requirements analysis is used to capture the business the software system is to be placed in. The data of the business is usually described by entity-relationship diagrams (possibly extended, e.g. class diagrams), the behaviour can be described by data-flow diagrams or event diagrams. The behaviour description very often will be based on some kind of organizational diagram of the enterprise. Since it references human activities, one should not aim at a complete behaviour specification capturing all possible models. The behaviour description should mirror the perspective of the business representative and the users on the business and the embedding of the software system to be build. It should help the system designer to identify the activities and the objectives relevant to the software system (consistency and formalizable properties being less significant). This cannot be directly supported by mathematical semantics [Blu93]. However, the formal semantics helps to define compact notations and techniques which in our view are missing in the area of business process modelling.
At this stage the distinction between data and behaviour is relevant. Most kind of data is already formalized to some extent (e.g. using specific forms) such that it seems feasible to strive for a complete specification of the data involved. Also, as discussed in [GZ92], data represents the stable part of the business which will have a direct counterpart in the software system. Therefore the effort necessary to obtain the complete specification will pay off in the rest of the project.
Altogether the following documents should be produced:

- a thorough data specification of the business (using the formal semantics to specify all kinds of integrity constraints),
- an exemplary behaviour description of the business which allows to identify the structure of the business activities and
- a thorough description of the objectives of the different participants.

3.3 Requirement Definition

Requirements definition identifies the data and the activities of the business to be incorporated into the software system. On one hand the possible system states are specified. This can be derived from the data specification documents of the requirements analysis. On the other hand the possible inputs and the system reaction in terms of data changes and output (called *system functions* in the following) have to be determined.
For definition of the system functions the perspective of the users is important. The only hard constraint is that consistency of system states has to be preserved by the system functions. The granularity of the functions (how many activities are covered by one system function) depends very much on the non-functional requirements and the perspective of the users. To allow for user participation in the development process the behaviour of the system functions must be specified using informal notations [3]. The formal semantics is necessary for validation of the design against the requirements definition.
Thus the following documents should be produced:

- a precise specification of the consistent system states,
- for each system function a precise specification of the input/output behaviour and the induced changes of the system states and
- a description of the structure of the user interface.

3.4 Logical System Design

Here the logical structure (architecture) of the software system is devised. The responsibility for data and the system functions has to be distributed over a set of (logical) components. This way the system behaviour in terms of component interaction is determined. To allow for a smooth development process it should be possible to view the documents of the requirements analysis and definition as abstractions of the logical system model.
Up to now the structure and style of the system has not be mentioned in the documents. So one can e.g. choose an object-oriented design or a more traditional module-oriented design. Also reuse of existing (software) components is important here. On the logical level a fine grained distribution of responsibilities is desirable allowing for centralization of functions in the physical design.
Clearly, the design of the structure is a difficult task which cannot be directly supported by formality. However, to allow for a thorough validation against the requirements definition, a precise semantics should be given to the design. Since the design will be developed incrementally, also the formal specification should be developed incrementally (for examples of suitable formalisms see [HSJ+94, PW94]).

[3] compare the different types of matrices employed in SSADM. Also prototyping will be helpful.

The logical design consists of

- a precise specification of the system components and their interaction

3.5 Versions

We have discussed how to integrate formal and informal documents within one system version. By revision all advantages of formality seem to be lost, since it might be difficult to relate the semantics of the old version to the semantics of the new version. However, this is also true for informal semantics. To allow for revision it is necessary to structure the documents, formal or informal, and to interrelate corresponding parts of different documents such that the revision can as far as possible be confined to parts of one version.

4 Conclusion

By differentiating the three specification perspectives *data, behaviour and objectives* and the three agreement perspectives *business,user and system* we have structured the development process into stages and documents with different emphasis on formality and participants. Only the system designer can take direct advantage of the formal semantics (mainly through suitable tools). Therefore formality should only be required for documents which can be related through formal techniques to the system design. There will always be a rich set of informal documents (referring to objectives and the business in general) whose relation to the system design is important, but can only informally be traced.

Acknowledgements

Thanks are due to the SysLab-group, in particular Alfred Aue, Manfred Broy, Eva Geisberger, Christoph Hoffmann and Jürgen Kazmeier, for critical discussions and remarks on the proposed methodology.

References

[BDD+92] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. Gritzner, and R. Weber. The design of distributed systems - an introduction to FOCUS. Technical Report TUM-I9202, Technische Universität München, 1992.

[Blu93] B.I. Blum. Formalism and prototyping in the software proccess. In *Program Verification (eds.: T.R. Colburn, J.H. Fetzer, T.L. Rankin)*, pages 213–238. Kluwer Academic Publishers, 1993.

[DCC92] E. Downs, P. Clare, and I. Coe. *Structured systems analysis and design method: application and context.* Prentice-Hall, 1992.

[FL94] M.J. Freeman and P.J. Layzell. A meta-model of information sytems to support reverse engineering. *Information and Software Technology*, 36(5):283–294, 1994.

[FRS89] C. Floyd, F. Reisin, and G. Schmidt. Steps to software development with users. In *ESEC 89, LNCS 387*. Springer Verlag, 1989.

[GZ92] G. Gryczan and H. Züllighoven. Objektorientierte Systementwicklung - Leitbild und Entwicklungsdokumente. *Informatik Spektrum*, (15):264–272, 1992.

[Het93] R. Hettler. On the translation of E/R schemata to SPECTRUM. Technical Report TUM-I9333, Technische Universität München, 1993.

[HSJ+94] T. Hartmann, G. Saake, R. Jungclaus, P. Hartel, and J. Kusch. Revised version of the modelling language TROLL. Technical Report 94-03, Technische Universität Braunschweig, 1994.

[Hus94] H. Hussmann. *Formal Foundations for SSADM.* handed in as Habilitationsschrift, Technische Universität München, 1994.

[JPRS94] M. Jarke, K. Pohl, C. Rolland, and J. Schmitt. Experience-based method evaluation and improvement. In *IFIP 8.1. CRIS Conference*. North-Holland, 1994.

[Nic94] F. Nickl. Ablaufspezifikation durch Datenflussdiagramme und Axiome. In *GI-Jahrestagung 1994, Informatik aktuell*, pages 10–18. Springer Verlag, 1994.

[OHM+88] T.W. Olle, J. Hagelstein, I.G. Macdonald, C. Rolland, H.G. Sol, F.J.M. van Assche, and A.A. Verijn-Stuart. *Information Systems Methodologies.* Addison-Wesley, 1988.

[Poh93] K. Pohl. The three dimensions of requirement engineering. In *CAiSE 1993, LNCS 685*, pages 275–292. Springer Verlag, 1993.

[PW94] P.Pepper and M. Wirsing. *KORSO: A Methodology for the development of correct software.* draft version, 1994.

[PWM93] F. Polack, M. Whiston, and K. Mander. The SAZ project: Integrating SSADM and Z. In *Formal Methods Europe 1993, LNCS 670*, pages 541–557. Springer Verlag, 1993.