

Constraint-Driven Software Design: An Escape From the Waterfall Model

Robert de Hoog
University of Amsterdam

Ton de Jong
University of Twente

Frits de Vries
University of Amsterdam

ABSTRACT

This paper presents the principles of a development methodology for software design. The methodology is based on a nonlinear, product-driven approach that integrates quality aspects. The principles are made more concrete in two examples: one for developing educational simulations and one for developing expert systems. It is shown that the flexibility needed for building

high quality systems leads to integrated development environments in which methodology, product and tools are closely attuned to each other. This "development process reengineering" can lead to significant improvements in the quality of the product in terms of both maintainability and performance enhancement of the people involved in the development process.

Introduction

Prescribing some kind of procedure is a frequently used way of improving quality and performance. It has been applied in many different areas, especially where construction of an artifact is the immediate concern. Constructing computer programs is clearly one of them. Prescribed procedures are often called "development methodologies" in this field. Why did development methodologies emerge during the late sixties and early seventies? Because the sheer size of the programs that had to be developed made it impossible to live any longer according to the old

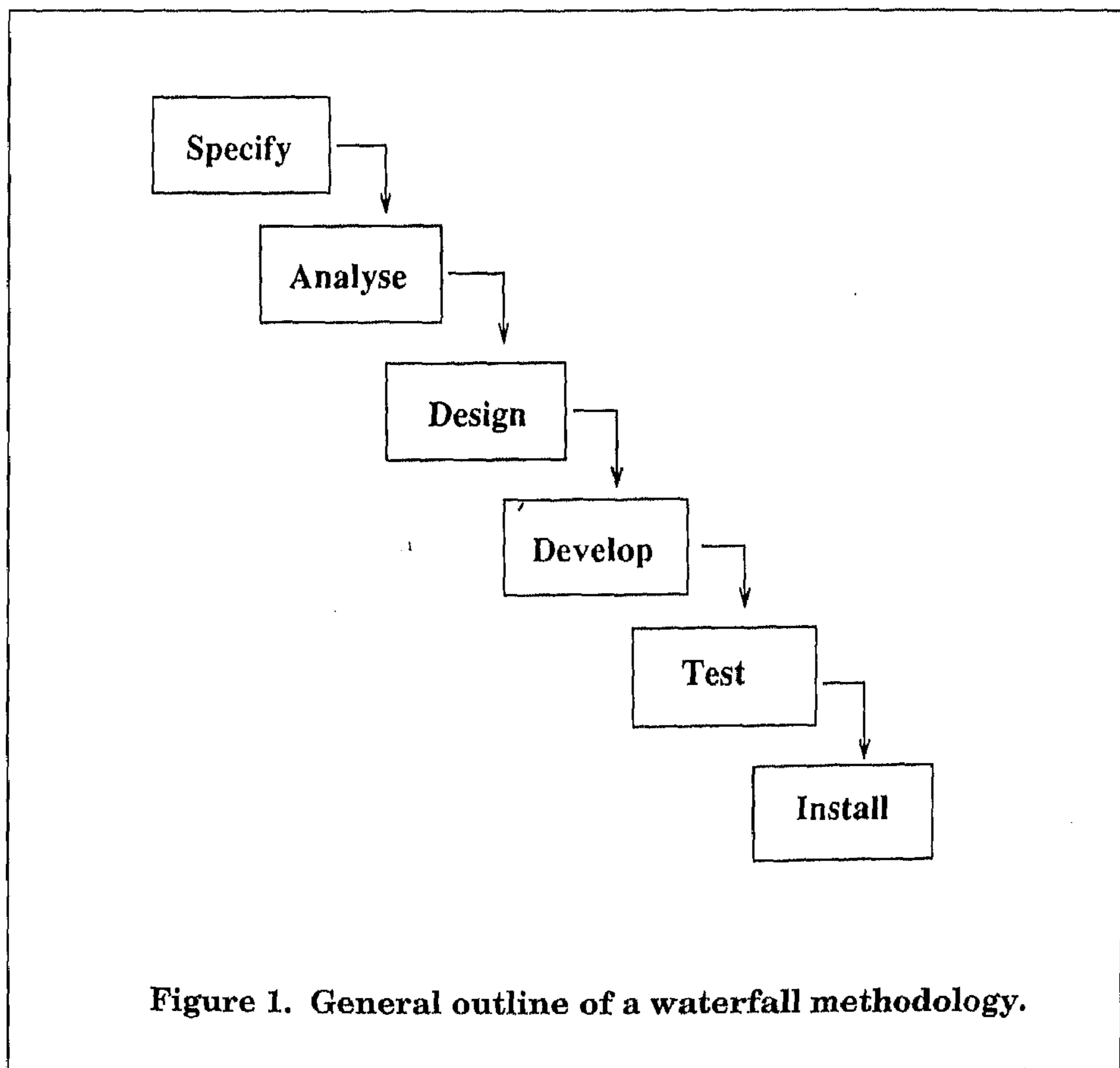
adage of the programmers, "Code first, design later (or not at all)." The resulting software never materialized, was produced far too late, or turned out to be impossible to maintain properly. Development methodologies forced programmers into adopting what one could call the "architect's approach." The most pervasive property of this approach is the sequence "analyse, design, build" which is still visible in all waterfall methodologies (see Figure 1). This approach is based on the hard fact that in architecture (as in software development) the destruction of a half-built structure due to incorrect

analysis and design is simply too costly. The linear and inflexible nature of traditional waterfall methodologies limits the possibilities for modifying the product during the development process. As a general implication, we can say that development methodology, product, and development tools are strongly intertwined. A more flexible development process must go hand in hand with an easily modifiable product and must be supported by a methodology and tools that assist in managing this flexibility. The goal of this paper is to present the principles of a methodology that satisfies these requirements. The use of the methodology

(and its associated tools) is demonstrated using two examples based on actual projects (authoring educational simulations and expert systems development). The claim is that methodologies and tools of this type will fit better into actual design practice and will contribute to a substantial improvement in the performance of the people involved in the development process as well as in the product itself.

Methodologies and Tools

In this section we will present a brief overview of principles underlying development methodologies currently in use and the present state of



Computer Aided Software Engineering (CASE) approaches. This will provide some general background information for later sections.

Waterfall methodologies

Development methodologies came into existence as a reaction against the "footloose and fancy-free" (Overmyer, 1990) styles of early programmers. The so-called "waterfall" model has been, until recently, the most frequently used model for controlling and guiding complex software development projects. The basic idea underlying this model is that development proceeds in stages. Each stage or phase must be finished in its entirety before a new phase can start. Just as water in a waterfall cannot flow back, phases that are finished should not be started again. The waterfall model has generated an avalanche of methodologies, which we will not describe in detail. Figure 1 gives a stylized representation of phases that occur, under one name or another, in almost all methodologies that are based on the waterfall concept.

Boehm's Spiral Model

Boehm's (1988) spiral model (see Figure 2) was developed as an alternative to standard waterfall models. The main problem Boehm tried to solve with his approach is the lack of flexibility to deal with project-specific risks in waterfall-based methodologies. These methodologies give the false impression that everything in a project will unfold as prescribed by the methodology, whereas in practice, changes and adjustments are more the rule than the exception. His spiral model reflects this preoccupation with risk. Each cycle in the spiral

(each full circle in Figure 2) starts with an assessment of the risks associated with possible next steps in the development work (the "Risk" quadrant). In general, Boehm advocates carrying out first the development activities with the highest associated risks. In his view, good project management means doing risky things first, because doing them later may entail far larger costs when something goes wrong. After selecting the riskiest activities, one has to plan them in a coherent way, just like planning activities in standard software project management (the "Plan" quadrant). In the "Development" quadrant, actual work is carried out on the product. Another crucial quadrant in Figure 2 is the "Review" quadrant. As Boehm states, reviewing measurable outcomes of activities lies at the heart of sound software development. No activity should be considered finished before the review of its outputs has been positive. This quality-driven aspect is a cornerstone of Boehm's work that is also espoused by other authors (e.g., DeMarco, 1982). Boehm's spiral is a powerful metaphor for thinking about software development. One criticism is that it lacks a set of detailed activities that are prescribed for actual software development work. Another criticism is that a definition of the axes on which the spiral unfolds is lacking. Obviously they seem to represent some kind of "gain" in the development process, but how this gain can be expressed in terms of undefined axes remains unclear. Also, combining product development and project management as implied by having the "development" quadrant together with the management-oriented quadrants (risk, plan,

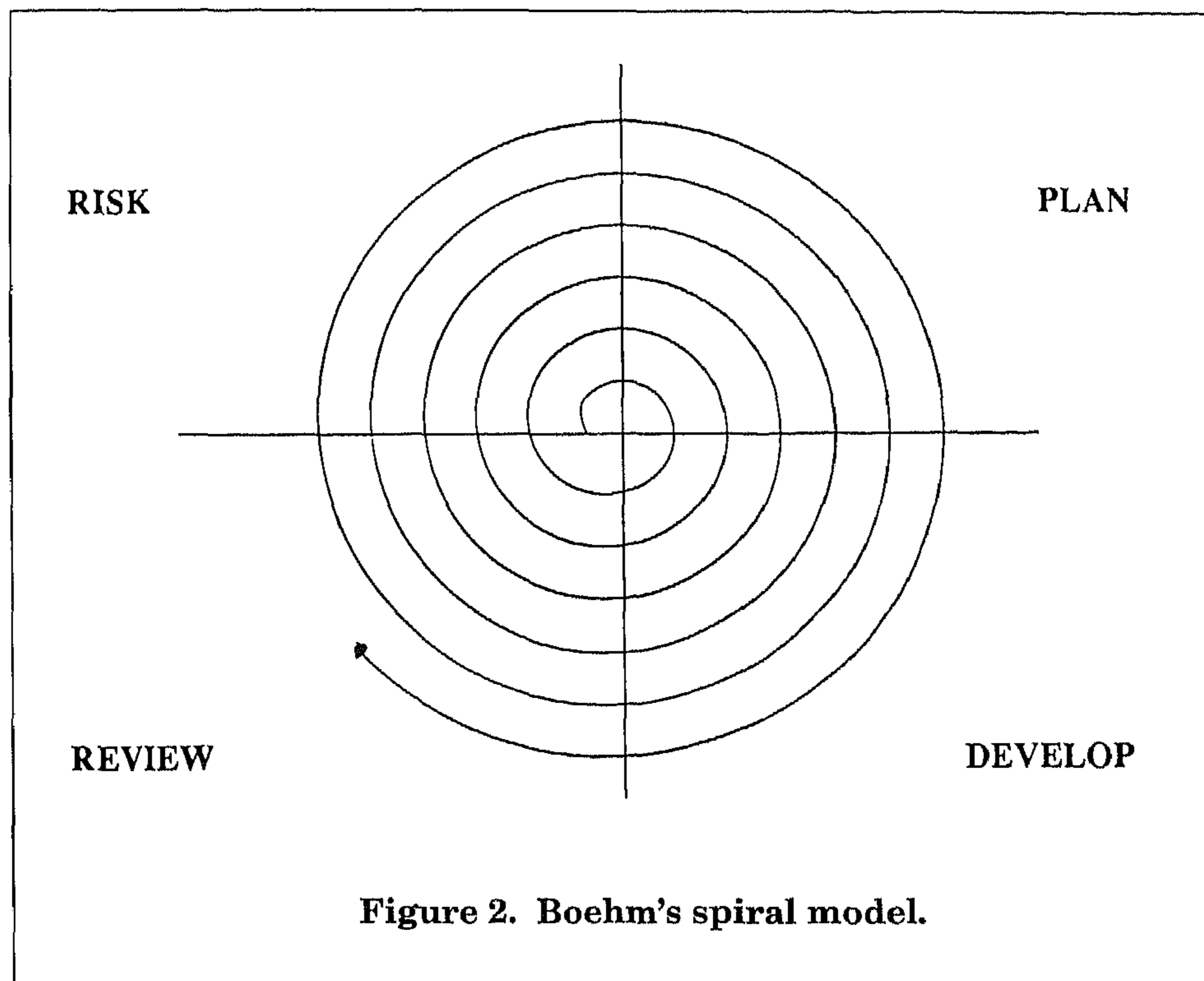


Figure 2. Boehm's spiral model.

review) can be questioned. Some authors (see for example Berkeley et al., 1990) hold that, at least for conceptual purposes, a rigid separation between system development and project management activities must be observed. By including the development quadrant, the spiral model mixes these two aspects.

Other authors have also tried to sidestep the waterfall approach. Glasson (1989), for example, describes a methodology (or "meta-model") based on states of products and deliverables that can be realized in a parallel and nonlinear fashion. The citation below aptly summarizes Glasson's viewpoint.

The meta-model sees information systems evolving as they are developed and used. It uses system

states to describe and control that evolutionary process. It uses deliverables, which are simply the outcomes of system-development work, to define a system as being in a particular state of evolution. Creating new deliverables causes a change of state.

(Glasson, 1989 p.351)

Meyer (1991) presents a "cluster approach" to system development that reiterates that parallel development of key elements of a system is feasible and even desirable. Some of the ideas that will be introduced in this article are based on the work of these authors. The examples of the methodology described here, however, go beyond prior work by giving clear-cut definitions of what products, deliverables, and states are.

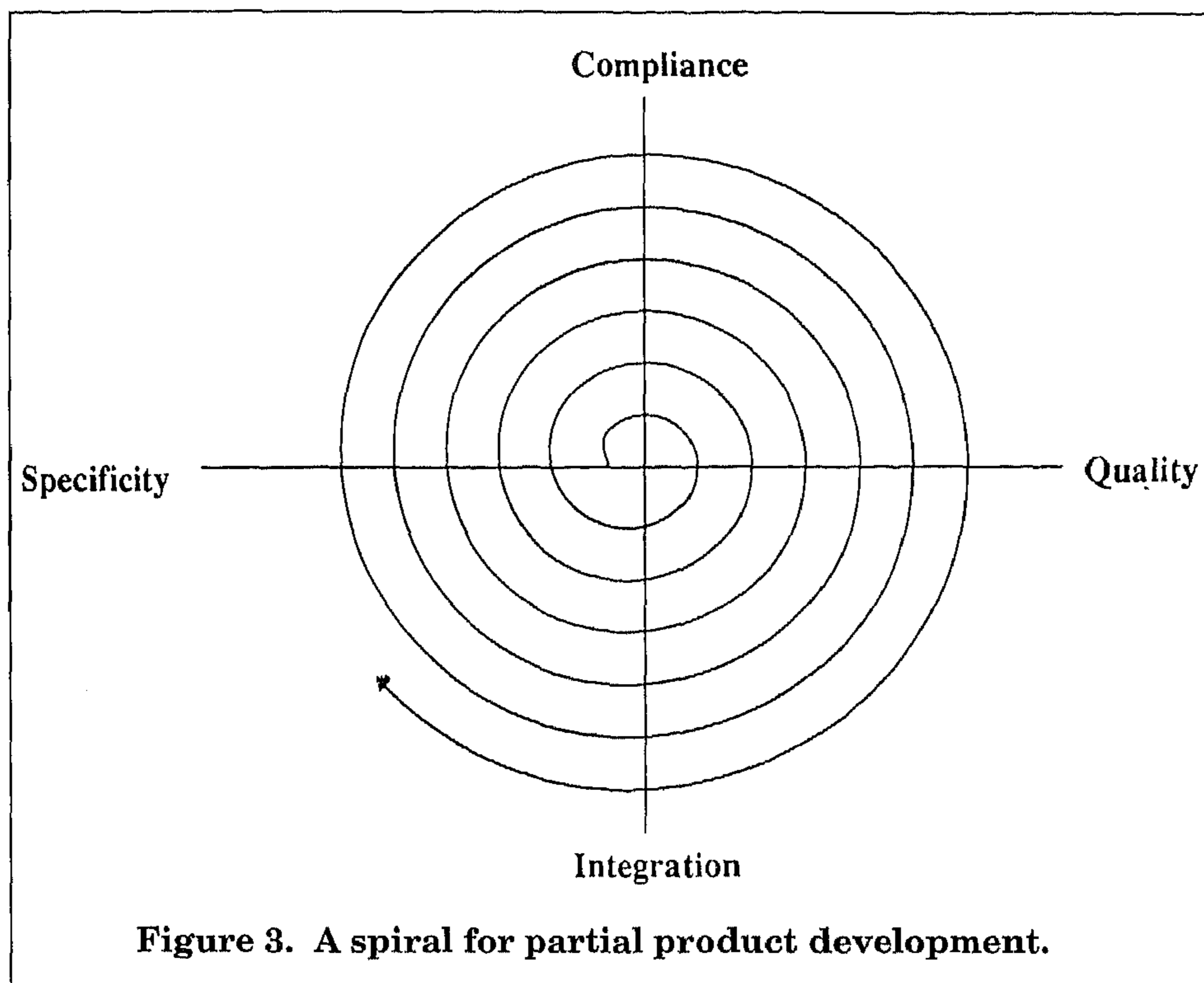
Tools

Methodology, product, and tools are inextricably intertwined. The notion of "tools" can be seen in workbenches for C(omputer) A(ided) S(oftware) E(ngineering). During the last decade, some significant advances have been made in the area of Computer Aided Software Engineering tools or "workbenches." For example, Software Development Environment (SDE) is an integrated set of CASE tools to assist the developers of a complex or large software system. The fundamental objective of a SDE is to automate the task of producing software (Garlan & Ilias, 1990). What kinds of tools are integrated in a development environment? Several hundred CASE tools are already commercially available. These can be grouped into categories of tools for: 1) enterprise modelling; 2) analysis and design; 3) software generation (3GL, 4GL); 4) testing and maintenance; 5) planning and project management; 6) user interface management; 7) building knowledge bases; 8) teamwork coordination; and 9) version management. The wide range of available tools will have the effect that what used to be paper and pencil work will be more and more often accomplished with a computerized tool assisting the software developer. Attempts to integrate these tools in an SDE or workbench leads to several important problems. Problems are related to:

- Coupling of separate tools (see, e.g., Garlan & Ilias, 1990)
- Decentralized software development (see, e.g., Smit, 1991)
- Prototyping as a development method (see, e.g., Mullin, 1990)
- Uniform representation mechanisms (see, e.g., Ward, 1990)

- Software development as teamwork (see, e.g., Noparstak, 1990)
- Workbenches and visualisation (see, e.g., Rubin et al., 1990)
- Methodological cohesion and integrity (see, e.g., Winblad et al., 1990)

This last problem is especially relevant for our considerations concerning the relationship between a workbench and a methodology. This problem follows from the requirement that given a certain development methodology, the integrated tools in the workbench must behave in a way consistent with that methodology, which also holds for imported, deleted, and interchanged tools. More specifically: how can prescriptions of a methodology be assimilated into integrated workbenches, when tools are related to methods? First, the architecture of the framework has to permit the import of new tools, it must be an "open" environment. Wybolt (1991) uses the term "framework" for a basic architecture consisting of services for presentation integration, an integration agent, integration services, and a repository where independent tools can be integrated. If such an architecture is more closed, the assimilation of new tools is hampered and the framework will be obsolete when new methods arise. Second, the coupling of the newly integrated tools can destroy the integrity of the methodology being used, when the imported new tool does not fit. For instance, tools for iterative design methodologies can hardly be replaced by tools based on the waterfall approach. As can be gauged from the remarks above, there ought to be a strong connection between the design of a workbench



and the design of a methodology. In the methodology proposed in this paper, we try to accomplish this tight integration by tailoring the methodology to the workbench and vice versa.

The Methodology

In this section, the basic elements of the proposed methodology are presented. We show that spiral development and quality assurance and quality control can be closely integrated. The methodology should facilitate parallel and nonlinear development of the software product. First, the product has to be decomposed into several partial products that can be built separately. Work on partial products is called *local* de-

velopment. Work on the whole process is called *global* development.

Local Development

The dynamics of the local development process are captured by the notion that for each partial product, a separate spiral is built which reflects the incremental development of the partial product under consideration. This spiral is depicted in Figure 3. As can be seen in Figure 3, the partial product is pushed along four axes, each axis depicting an important criterion for measuring progress in building the partial product. What the spiral "says" can be summarized as follows: In building your partial product, you start by trying to make it more specific. If you have achieved this, you check whether you have

made it more specific in the right direction, that is, whether the enhancement is consistent with the previous state of the partial product and maintains consistency and coherence with other partial products. The axes in Figure 3 will be described in more detail. The case descriptions provide examples.

Specificity. The idea behind this axis is that it summarizes the actual (next) state of the partial product as realized by the current activity. By specificity we mean that each subsequent partial product state must be an improvement over the previous one in terms of being more specific. One could imagine that every partial product can be described as a frame with slots that have to be filled during the development process. The more slots are filled, the more specific the partial product becomes. Every configuration of filled slots can be seen as a specific state of the partial product. This view of partial products, as frames and slots to be filled during development, permits in principle the management of the development process. Milestone management can be linked to the occurrence of certain states of the partial product which are deemed crucial. Thus development activities result in crucial partial product states as milestones that can be monitored with traditional project management methods and techniques.

Compliance. Compliance means that the development of the partial product does not contradict what is specified in external requirements for the partial product.

Quality. This axis measures progress in terms of the quality of the outcome of the development step taken. Does the activity undertaken

in the current loop of the spiral result in a new state of the partial product that is still consistent with its previous state and eventually its foreseeable future states? Though we feel that for the time being this is an adequate description of what we mean by quality, there is no reason in principle why other, more conventional notions about quality and quality assurance and control cannot be accommodated by this axis.

Integration. Just as the quality axis refers to consistency within a partial product, the integration axis takes care of consistency and coherence between different partial products. Are states of different partial products mutually consistent? In other words: every activity's results must be checked against the current states of other partial products. When an inconsistency or a conflict has been identified a solution must be found. This integration axis connects the different local spirals for the partial products to the global development of the product as a whole.

It must be emphasized that the spiral in Figure 3 is stylized. There is no intrinsic need for the "intervals" on each axis to always be the same. It is perfectly possible that for some time more gain is achieved on one axis than on another. For example, in the beginning of an activity one could make considerable gains in specificity and quality, with compliance and integration lagging. Thus, the spiral can become lopsided. If this continues for *too* long, however, this is a sign that there are problems in the development process.

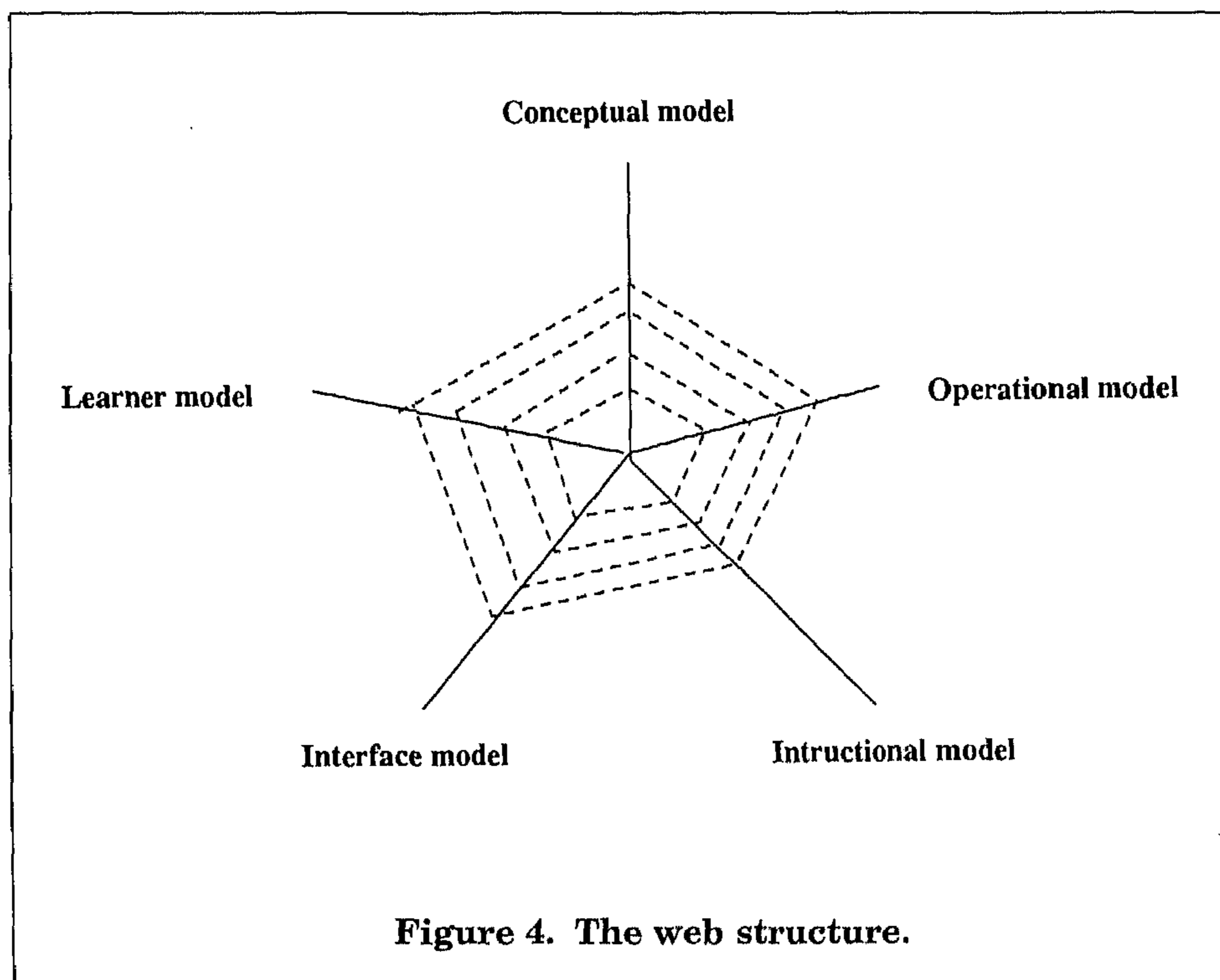
Global development

The consequence of the local development spirals is that there is no

longer a necessity for a strong, phase-driven development approach for the whole process (as in all waterfall methodologies). In our approach it will be possible that different partial products are in different states of development. Next we will present an example of a project for the development of a software product, a Multimedia Integrated Simulation Learning Environment (MISLE) that consists of different "models." Our definition of a "model" is very general: a structure that consists of a specified number of elements and well-defined relationships between those elements. This covers in principle the whole range from an equation to scaled down and stylized structures used in architecture. These models are regarded as the partial products and each of these parts (models) can

be in a different state of development. The global development level (the development state of all partial products) can be visualized as a kind of web structure.

Figure 4 is a web-like structure in which each axis represents the development of one of the models from the MISLE. In the development process, partial products ("models") are "pushed" outward, meaning that they come closer to their "desired" state. The development of the models does not start from the center, because in the workbench, authors using MISLE have access to a library of predefined templates for model elements. They can use these in developing the models. It is an essential characteristic of the methodology that at a certain point in time not all products need to be the same distance



from the center, reflecting the nonlinear character of the methodology.

Local and global development can be shown simultaneously by drawing a picture that combines Figures 3 and 4 (see Figure 5). In Figure 5 the local spiral for each model unwinds itself on the axis in Figure 4 associated with the model.

In the next two sections we will illustrate the principles underlying the methodology by means of two cases in which more specific instances of the methodology have been developed.

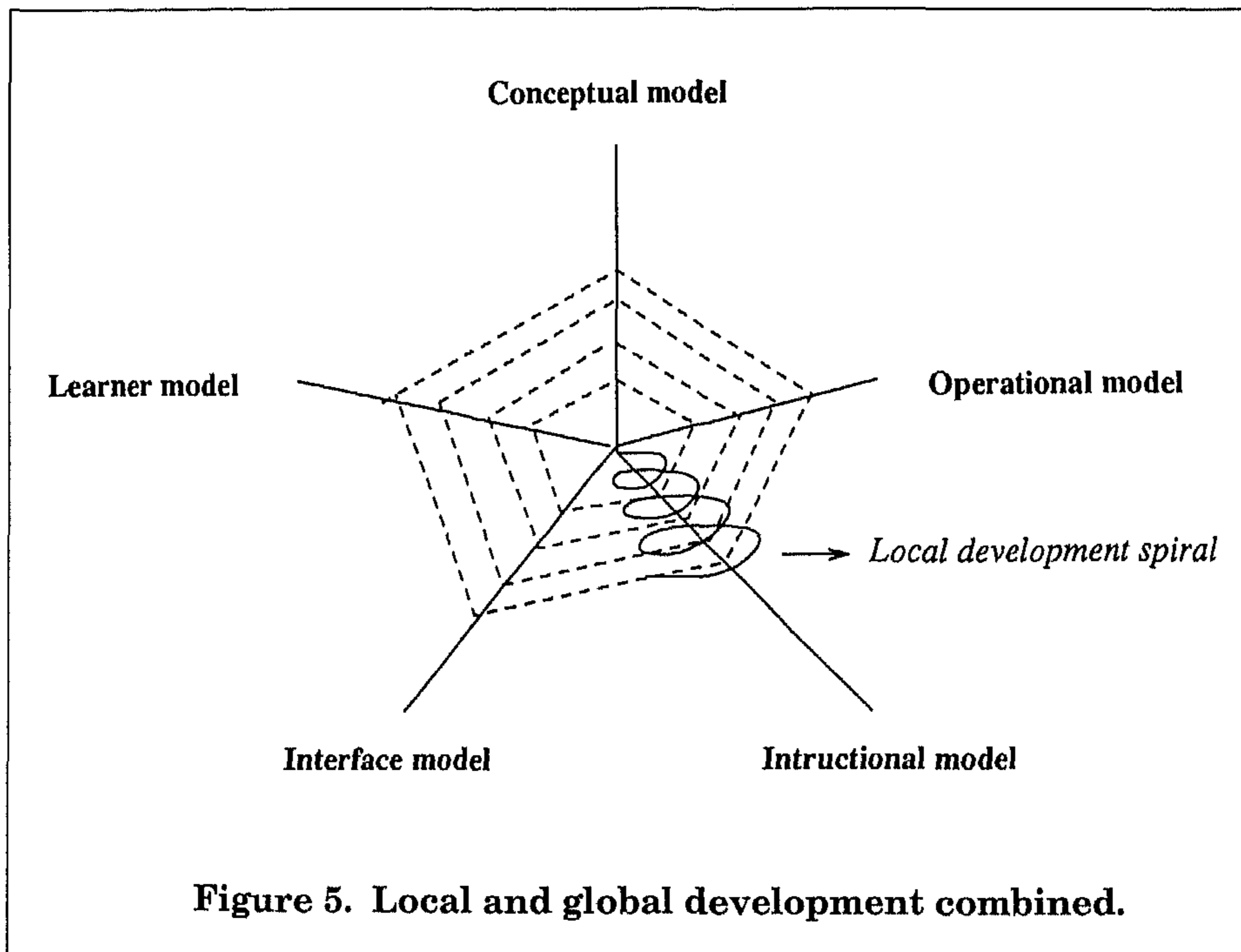
The SMISLE Case

The methodology outlined previously is made more specific in the context of the SIMULATE and SMISLE projects. Both projects had as their aim the development of an authoring environment for computer

simulations in an instructional context. The development task to be supported by the methodology is authoring of educational computer simulations. The partial products are models; the methodology deals with the controlled construction of models. Extensive overviews of these models are given in de Jong, Tait, & van Joolingen (1992), and in van Joolingen & de Jong (1991). Here we will give a short summary taken from de Jong et al. (1994).

The cognitive model

The cognitive model contains domain descriptions (representations of the domain the simulation deals with, for example, a simple harmonic oscillator) that are tailored to giving instruction about the domain. These descriptions can either refer to operational domains or to conceptual domains, thus leading to an operational



model or a conceptual model (see Figure 4). These two models represent what we want the learner to know or master at the end of the training session (see also van Joolingen & de Jong, 1992).

For those with a software design background, a "runnable" version will be derived from this cognitive model. This "runnable" version has as its objective to enable an optimal, fast, and efficient simulation. It will, therefore, often have a numerical character, but it may also be a qualitative, causal model.

The Instructional Model

This model contains the instructional function of the simulation. It carries all the instructional measures available for the learner, such as the possibility of presenting explanations, formulating hypotheses, carrying out specific assignments, and so forth.

The Learner Model

This model can store (or infer) information regarding a learner. This information can then be used for activating instructional measures.

The Interface Model

This model takes care of the interface between the simulation and the learner. It contains all kinds of graphical objects, windows, and so forth, enabling the learner to interact with the simulation.

These four models, or partial products, are in line with a division into components that is commonly used in describing intelligent tutoring systems (see for example Wenger, 1987).

Authoring with the SMISLE authoring system (built in Smalltalk and Visualworks) essentially means building an instructional simulation based on building blocks from libraries

of building blocks offered by SMISLE. For each model, a library of building blocks is available for the author. Examples of building blocks are templates for instructional measures such as assignments and explanations. (Note that only the runnable model is not created directly by the author but is generated from the cognitive model.) The author has to develop five different models (see Figure 4). The development of these models proceeds in an iterative fashion that depends on the nature of the domain and the preferences of the author. The authoring methodology supports development by guiding the progression along the axes in Figure 3. We will give examples of this for each of the axes below.

Specificity

In the SMISLE authoring environment, making a model more specific amounts to selecting a building block from the library available for the model under consideration. For example, when working on the interface model the author can select a window type from the interface library. As another example, assume the author chooses to work on the instructional model. Then he or she can select an assignment type from the instructional model library. The methodology guides this selection process by graying out options that are not admissible.

Compliance Requirements fall into two main groups:

Educational Environment Requirements. The educational software that must be developed has to function in a specific educational setting. This setting is the source of constraints on model development.

Examples of factors that are relevant in the educational environment are school policies, instructional prerequisites, and other available educational materials. How to arrive at these requirements is adequately described in existing methodologies that can serve as an aid to obtaining them in the SMISLE context. We simply assume that they are known

to the author either through the workbench or through any other medium. Furthermore, we assume that, given the domain-specific

nature of these requirements, linking them to models is the task of the author.

Hardware and Software Requirements. This aspect addresses the hardware and software requirements for the target system. Examples of factors that are relevant are speed of available computers, storage capabilities, nature of the display medium, and other available electronic media. More factors can be found in de Hoog et al. (1991) who discuss hardware requirements for simulation programs. Again we assume that the collection of these requirements can be done with existing techniques and that linking them to models is up to the author.

Quality

As an example of how quality impacts design, assume that an author has decided on progressive implementation as the best instructional

strategy. This strategy says that the domain description and its salient, relational features must be discovered gradually (in phases or steps) by the student. Through this decision the author has pushed the instructional model into a well-defined state ("progressive implementation chosen"). In further developing the instructional model, s/he must take

care that additional modelling steps stay in line with this state. Defining an instructional tactic that shows the learner the complete

domain description in an early stage of the simulation session(s) clearly violates the consistency requirement set by the quality axis of the spiral. By satisfying the quality axis, the author guarantees that the model being developed stays internally consistent.

Integration

Consider for example the situation in which the interface model is to be built (e.g., the manipulation possibilities for the learner). If the instructional model calls for goal decomposition by the learner, one has to check whether the designed interface model does not contradict what has been stated in the instructional model. If the interface model does not enable the learner to carry out goal decomposition, this is inconsistent with the current state of the instructional model. The interface model is

Good project management means doing risky things first, because doing them later may entail far larger costs when something goes wrong.

then externally inconsistent (with other models).

It must be emphasized that there is not an *a priori* preferred way of developing the models in a fixed sequence. We have observed authors starting with the interface model, while others worked first on the cognitive model. The methodology as implemented in the workbench supports these different ways of working by providing a general task tree that can be traversed by the author. Also, the methodology keeps track of the four axes described above by monitoring the selection-instantiation-integration cycle for the different models. A more detailed description of this facility can be found in Kuyper et al. (1993).

To conclude this section we can state that the SMISLE development environment provides a good example of how a flexible development process (creating and modifying models in a nonlinear way) can be supported by a tailored methodology and integrated tools (building blocks and workbench).

The CommonKADS Case

The CommonKADS methodology is designed for supporting the systematic development and maintenance of expert systems. It is characterized by the principles outlined: a spiral model, driven by product decomposition (models), driven by quality and driven by risk. Due to the generally risky nature of expert system development projects, there is a stronger emphasis in CommonKADS on risk analysis and risk reduction than in SMISLE. However, just as in the SMISLE case, the partial products of a CommonKADS project are models. CommonKADS has six dif-

ferent models that are described here briefly.

The *Organisation Model* captures organisational features that can influence expert system development (e.g., impacts on the organisation, knowledge bottlenecks in the organisation).

The *Task Model* gives the task decomposition that is the focus of the expert system development (e.g., the general task of decision making in the field of medicine can be decomposed into subtasks such as "intake of patient," "diagnosis of patient").

The *Agent Model* represents the features of every agent (for example, a computer program) that plays a role in performing the current and future tasks (e.g., an agent is not capable of obtaining sensory data).

The *Expertise Model* describes the expertise the agents need to carry out their tasks (e.g., the knowledge needed to be able to make a correct diagnosis).

The *Communication Model* describes the communication between the agents involved in carrying out the tasks.

The *Design Model* represents design decisions made during the expert development process (e.g., type of computer platform, programming environment chosen).

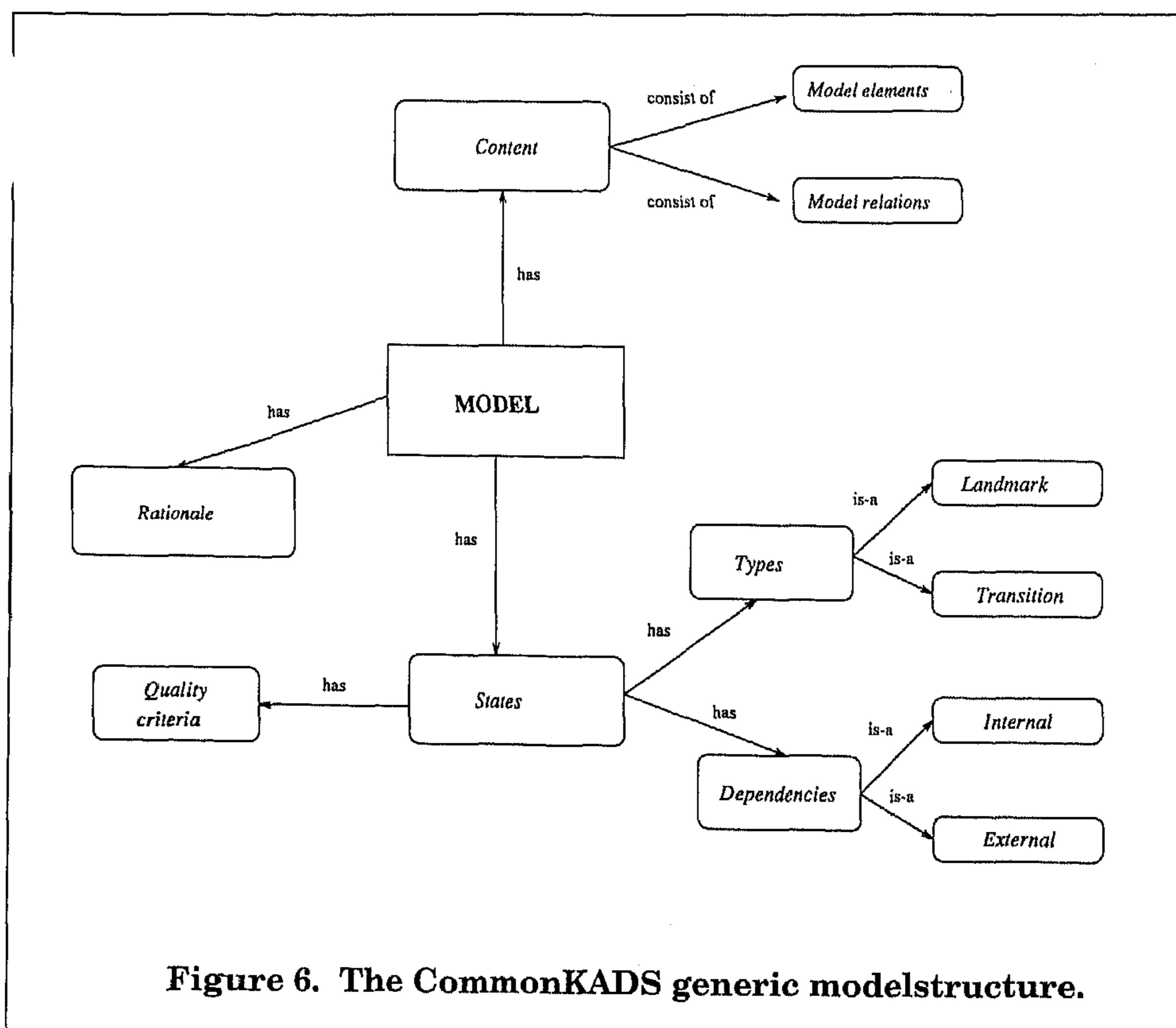
All of the models are seen as equally important products of the project; they all have to be developed. The various models can be seen as partial products that go through different development stages. These stages are model states. Attached to each model is a model-specific set of states that represent the different development stages the model can be in. A model state can be characterized as a well-defined, observable condition the existence of which can

be unambiguously established. The project plan is the set of all model states relevant for the project scheduled over the time available for completion.

Apart from determining whether a model is in a certain state or not, it is often also very important to measure the quality of that state. This again brings in the issue of quality assurance and quality control. The quality of a state can be measured against a set of quality criteria that can, but need not be, unique for each state. For example, an important criterion is whether requirements have been met or not. Thus, model evolution can also be seen as a gain in

terms of a number of quality axes, just like the outward moving cobweb in Figure 4. A model in CommonKADS has a generic structure that is shown in Figure 6 (it should be noted that this generic structure satisfies the general definition of a "model" given earlier).

In Figure 6, the *content* refers to what is going to be put into the model. For all models mentioned above, a ready-made template is defined (see de Hoog et al., 1992) that gives in an abstract way the elements and the relationships between them that must be established during an actual project. For the expertise model for example, this means that the rel-



evant domain concepts must be described and related to the inferences that make use of them. The states are used for controlling the development of the models, just as in SMISLE. Quality criteria are applied to each state, thus taking care of the quality axis in Figure 3.

A CommonKADS project will consist of the development of the different models into their required states. The development process is driven by the risk-reduction approach advocated by Boehm. Model states that must be achieved during a particular cycle are largely derived from risks that must be addressed in that cycle. The methodology supports model development by providing the "building blocks" (the predefined model templates) and guidance for assessing risks, deriving project activities, and defining quality criteria. The cornerstone is, again, the ability to tailor the development process and the development methodology to the actual situation of a project. Models as partial products are recognizable and modifiable constructs that will persist throughout the "life cycle" of the product, not just during the development process. This explicit persistence of models enables in principle a better maintainability by providing partial products that are well defined.

The CommonKADS methodology is embodied in a (prototype) workbench that aims at comprehensive support of the process of building expert systems. It achieves this support not by forcing every project into the straitjacket of a waterfall model, but by providing the means for the project manager to configure the methodology according to the requirements of each specific project without sacrific-

ing the need for controlled and quality-driven project work. From initial validation exercises carried out in the KADS-II project (see for example Bauer & Strasser 1993; Pinedo, Raya & Gala, 1993) it has become clear that this methodology fits neatly into actual practice, thus enhancing and improving the quality of the work.

Conclusions

In this paper we have outlined the principles of a flexible development methodology. These principles have been incorporated into two methodologies: the SMISLE approach and the CommonKADS methodology. It has been shown that the concepts are useful for improving the quality of the work and the performance of the people involved. At the same time however, it has become clear again that the intertwining of methodology, product, and tools requires a careful and comprehensive approach. Though we are a bit reluctant to use fashionable phrases, it seems to amount to a kind of "business process reengineering." We substitute "development process" for "business process." A new methodology without a redefinition of the product and without the tools to deal responsibly with the increased flexibility, will probably do more harm than good. Improving and refining this "three-sided coin" will be high on the research agenda for the coming period.

References

- Bauer, C. & Strasser, A. (1993). An organisation model for Private Communication System Service. KADS-II/V3/WP/SIEM/003/1.0.
- Berkeley, D., de Hoog, R., & Humphreys, P. (1990). Software Development Project Management: Process and Support. Ellis Horwood.

- Boehm, B. W. (1988). A spiral model of software development and enhancement. *IEEE Computer*, 21(5), 61-72.
- DeMarco, T. (1982). *Controlling software projects*. Yourdon Press.
- Garlan, D., & Ilias, E. (1990). Low-cost, adaptable tool integration policies for integrated environments. *SIGSOFT Software Engineering Notes*, 15(6), 1-10.
- Glasson, B. C. (1989). Model of System Evolution. *Information and Software Technology*, 31(7), 351-356.
- de Hoog, R., de Jong, T., & de Vries, F. (1991). "Interfaces for instructional use of simulations". *Education & Computing*, 6(3,4), 359-385.
- de Hoog, R., Martil, R., Wielinga, B. J., Taylor, R., Bright, C., & van de Velde, W. (1992). The CommonKADS model set. Deliverable ESPRIT project P5248, KADS-II/WP I-II/UV/A/018/4.0, University of Amsterdam.
- de Jong, T., van Joolingen, W. R., de Hoog, R., Lapied, L., Scott, D., & Valent, R. (1994). SMISLE: System for Multimedia Integrated Simulation Learning Environments. In T. de Jong & L. Sarti (Eds.) *Design and production of multimedia and simulation based training material*. Kluwer, Academic Publishers.
- de Jong, T., Tait, K., & van Joolingen, W. R. (1992). Authoring for intelligent simulation based instruction: A model based approach. In S. A. Cerri & J. Whiting (Eds.) *Learning Technology in the European Communities* (pp. 619-637). Dordrecht: Kluwer Academic Publishers.
- van Joolingen, W., & de Jong, T. (1991). "An instruction related domain representation for simulations". In de Jong, T. & Tait, K. (Eds) *Towards formalising the components of an intelligent simulation learning environment* (pp. 17-65). Eindhoven University of Technology: DELTA project SAFE (P7061), Report SIM/19-20/A.
- Kuyper, M., Bredeweg, B., de Hoog, R. & van der Hulst, A. (1993). Authoring methodology. Deliverable D16, DELTA Project D2007 SMISLE, University of Amsterdam.
- Meyer, B. (1990). The new culture of software development. *Journal of Object Oriented Programming*, 3(5), 76-81.
- Mullin, M. (1990). *Rapid prototyping for object-oriented systems*. Addison-Wesley.
- Noparstak, B. (Ed) (1990). Special Expanded Tool Issue. *Scoop Smalltalk/V*, 3(4).
- Overmyer, S.P. (1990). The impact of DoD-Std-2167A on iterative design methodologies: help or hinder? *ACM Sigsoft Software Engineering Notes*, 15(5), 49-59.
- Pinedo, F., Raya, A. & Gala, S. (1993). Overall process validation: ONT Project. KADS-II/V3/TR/E/011/V1.0, ESPRIT Project P5248, Eritel, Spain.
- Rubin, R. V., Walker II, J. & Golin, E. J. (1990). Early Experiences with the Visual Programmer's WorkBench. *IEEE Transactions on Software Engineering*, 16(10), 1107-1120.
- Smit, Chr. G. (1991). Decentralisatie van systeemontwikkeling. *Computable*, 25 January 1991. (Translation: "Decentralizing systems development").
- Ward, P. T. (1990). The use of current-generation CASE tools in object-oriented analysis and design. In Spurr, K. & Layzell, P. (Eds.) *CASE on trial* (pp. 105-123). John Wiley & Sons.
- Wenger, E. (1988). *Artificial Intelligence and Tutoring Systems*. Morgan Kaufman.
- Winblad, A. L., Edwards, S. D., & King, D. R. (1990). *Object-oriented software*. Addison-Wesley.
- Wybolt, N. (1991). Perspectives on CASE tool integration. *ACM Software Engineering Notes*, 16(3), 56-60.

Authors' Note. The SIMULATE project was part of the SAFE project, partially funded by the CEC under contract P7061 (D1014) within the exploratory action of the DELTA programme. The SMISLE project is partially funded by the CEC as project D2007. Partners in this project are Framentec (F), ESI (F), Marconi

Simulation (UK), University of Twente (NL) and University of Amsterdam (NL). For an extensive description of SMISLE, see de Jong et al. (1994).

The work reported on the CommonKADS case has been carried out within the framework of the KADS2 project, partially funded by the ESPRIT Programme of the Commission of the European Communities as project number 5248. The partners in this project are Cap Gemini Innovation (F), Cap Programator (S), Netherlands Energy Research Foundation ECN (NL), ENTEL SA (ESP), Lloyd's Register (UK), Swedish Institute of Computer Science (S), Siemens AG (D), Touche Ross MC (UK), University of Amsterdam (NL), and Free University of Brussels (B).

ROBERT DE HOOG is Associate Professor of Social Science Informatics at the University of Amsterdam, Faculty of Psychology. His main research interests are in knowledge based systems and decision support systems. He has published over 50 papers in these fields and also co-authored a book on project management. He was and is involved in several international research projects sponsored by the European Community. *Mailing address:* University of Amsterdam, Faculty of Psychology, Dept. of Social Science Informatics, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands. E-mail: dehoog@swi.psy.uva.nl

TON DE JONG studied cognitive psychology at the University of Amsterdam and received a PhD from the Eindhoven University of Technology on the topic 'problem solving and knowledge representation in physics for novice students'. After his studies, he joined Delft University of Technology and became researcher and senior researcher at both the University of Amsterdam and the Eindhoven University of Technology. Since 1992, he has been Associate Professor at the University of Twente, Faculty of Educational Science and Technology. His main research areas are discovery learning with computer simulations and problem solving in physics. *Mailing address:* University of Twente, Faculty of Educational Science and Technology, PO Box 217, 7500 AE, Enschede, The Netherlands. E-mail: jong@edte.utwente.nl

FRITS DE VRIES teaches object orientation, databases, and human-computer interaction to Social Science students of the University of Amsterdam. His current interests include the envisioning of information, and dual coding theory. Dr. de Vries received a degree in sociology at the University of Amsterdam, and is a member of the ACM. *Mailing address:* University of Amsterdam, Faculty of Psychology, Dept. of Social Science Informatics, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands. E-mail: devries@swi.psy.uva.nl