



UNIVERSITÄT
KOBLENZ · LANDAU



Institut für
Wirtschaftsinformatik

Fachbereich Informatik
Universität Koblenz-Landau

MICHAEL PRASSE
PETER RITTGEN

SUCCESS FACTORS AND FUTURE CHALLENGES FOR THE DEVELOPMENT OF OBJECT ORIENTATION

Januar 2000



UNIVERSITÄT
KOBLENZ · LANDAU



**Institut für
Wirtschaftsinformatik**

Fachbereich Informatik
Universität Koblenz-Landau

MICHAEL PRASSE
PETER RITTGEN

SUCCESS FACTORS AND FUTURE CHALLENGES FOR THE DEVELOPMENT OF OBJECT ORIENTATION

Januar 2000

Die Arbeitsberichte des Instituts für Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i.d.R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The "Arbeitsberichte des Instituts für Wirtschaftsinformatik" comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

Alle Rechte vorbehalten. Insbesondere die der Übersetzung, des Nachdruckes, des Vortrags, der Entnahme von Abbildungen und Tabellen - auch bei nur auszugsweiser Verwertung.

All rights reserved. No part of this report may be reproduced by any means, or translated.

Anschrift der Verfasser
Address of the authors:

Michael Prasse
Dr. Peter Rittgen
Universität Koblenz-Landau
Rheinau 1, D-56075 Koblenz

**Arbeitsberichte des Instituts für
Wirtschaftsinformatik**
Herausgegeben von / Edited by:

Prof. Dr. Ulrich Frank
Prof. Dr. J. Felix Hampe

©IWI 99

Bezugsquelle / Source of Supply:

Institut für Wirtschaftsinformatik
Universität Koblenz-Landau
Rheinau 1
56075 Koblenz

Tel.: 0261-287-2520
Fax: 0261-287-2521
Email: iwi@uni-koblenz.de
WWW: <http://www.uni-koblenz.de/~iwi>



**Institut für
Wirtschaftsinformatik**

Fachbereich Informatik
Universität Koblenz-Landau

Abstract

Apart from the development of object orientation, a number of important technologies with considerable impact on software engineering have been introduced: concurrency, distribution, databases and formal techniques. In this paper we will show why these techniques should be integrated with object orientation to meet future requirements of software engineering.

Introduction

Today it is generally accepted that the introduction of object orientation was an important step towards a notable improvement of software development techniques and processes ([Pres1997], pp. 551 ff.). Considering that the major ideas on object orientation were already developed in the 70's and 80's one may ask where the next revolutionary improvements might stem from ([Kay1993], p. 40).

If we look closer at object orientation itself we recognize that it was rather the product of a continuous development than a sudden breakthrough. Some of the roots which took part in this development were the language Simula, which is sometimes called the first object-oriented language ([DMN1970]), and the idea of modularization and information hiding ([Parn1972]). From this point of view object orientation can be seen as one step in the evolution. But rather than treating object orientation as a separate issue we should see it in the appropriate context. During the development of Smalltalk one of the main ideas was that of personal computing. The starting point was Kay's version of a computing device which was small, mobile and can be used by everyone for managing textual, graphical or multimedia information. He called this device „Dynabook“. To handle these information objects he suggested Smalltalk, a language to express and model necessary real world artefacts ([Kay1993], pp. 2 ff.; [Gold1997], pp. 51 f.). Such ideas represented important progress because they led to the introduction of high-resolution monitors, the invention of graphical user interfaces and the development of user-centered applications and class libraries which made computers easy to use even for the layman.

If we analyze the situation today we find many similarities to that of the 70's and 80's. Globalization of distributed computer and information nets poses an important challenge to software-development techniques and will change our understanding of network applications rapidly. Therefore the future development of object orientation should not be considered on its own but as a part of the whole software cycle. Especially the increasing importance of additional requirements coming from concurrency, distribution, reuse and new application domains will dominate this development.

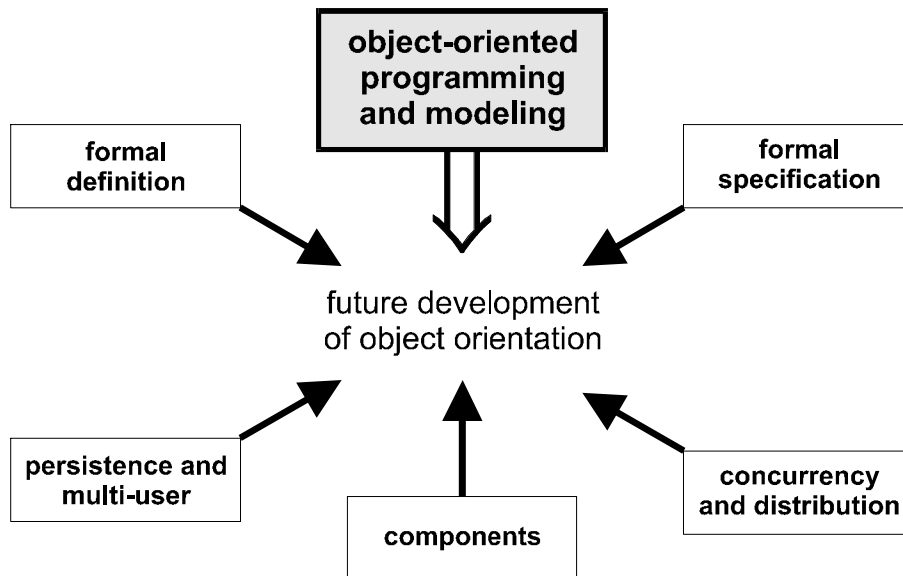
One possibility to look at the future of object orientation is to put the improvement of the software development process itself in the center of our attention. Here we focus on the technical perspective on object orientation. But we will not consider such approaches like subject-oriented programming ([HaOs1993]), adaptive programming ([Lieb1996]) or aspect-oriented programming ([KLM+1997]) which extend object-oriented concepts but have not yet reached a mature state. Instead we will investigate well-known factors or major trends and will argue that their integration with object-orientation offers a great potential, which is typically neglected in the literature. In the words of Tsichritzis and Nierstraß:

“With the gradual introduction and acceptance of object-oriented techniques, we find ourselves forced to pursue several directions of research along traditional lines, in order that we may understand how objects require us to reinterpret established results.” ([TsNi1989], p. 524)

The structure of the paper is as follows. First we give an overview over the success factors which we consider to be relevant for the future development of object orientation. Then we investigate the individual factors separately.

Overview over the Success Factors

To judge the future trends in object-oriented software engineering we can identify six factors governing the success of software projects from a technical point of view.



Technical Success Factors of Object Orientation

Object-Oriented Modeling and Programming (1)

This factor describes the core part subsuming today's understanding of object orientation. Object-oriented modeling and programming and the deployment of object-oriented technologies has become a dominating factor in software engineering. It is now an established standard for developing application software ([Pres1997], p. 551 f.). It includes common object-oriented programming languages such as C++, Eiffel, Java and Smalltalk ([Salu98]), also modeling languages such as UML ([OMG1999]) and OML ([FHGP1998]) and those used in the Booch-method ([Booc94]) and OMT ([RBPL1991]). It further includes development methods and processes such as the Unified Software Development Process ([JBR1999]) and OPEN ([HGY1997]), object-oriented testing and object-oriented design techniques such as frameworks and patterns ([LRP+1995], pp. 34 ff.; [GHJV1995]). Although some criticism and open problems remain, its wide-spread use is the prerequisite for future development.

Formalization of Basic Object-Oriented Concepts (2)

The second factor relates to the formal basis of the object-oriented paradigm. Currently almost any programming or modeling language uses its own interpretation of object-oriented concepts. In addition the semantics of some concepts such as inheritance has not yet been defined uniquely, which again leads to a multitude of interpretations. A common basis does not exist. To adapt to this situation many object-oriented modeling languages define their concepts only vaguely or recommend the use of the definitions of the intended programming language.

In the theory of programming languages and type systems intensive research is going on concerning a formalization of object orientation. Approaches are mostly based on a functional object-oriented paradigm neglecting the states of objects and they use lambda calculi ([CaWe1985]) or object calculi ([AbCa1996]). They helped in understanding the semantics of object, method, class, inheritance or polymorphism. They also led to the development of advanced polymorphic type systems for object-oriented languages. But at the same time they also point out current problems. For example, there are a number of questions regarding aliasing, inheritance or polymorphism which have not been answered satisfactorily (cf. [BCC+1995]). Furthermore, these approaches do not take into account practical considerations to a sufficient degree: often functional language models are used although the common modeling and programming languages are imperative.

Nevertheless, object calculi and similar formalisms can help in creating a formally founded and unified basis for object orientation which also respects software-engineering aspects.

Formal Specifications (3)

Formal specifications represent the third factor. Their value for the software-development process is generally recognized ([LaHa1994b], p. xiii). Nevertheless they are scarcely employed in practice. The reasons for this are manifold: higher development costs, lack of qualified personnel, occasional lack of quality concern and shortcomings of the current formal specification techniques.

Moreover the integration of object orientation and formal specification still presents a challenge. In many cases formal specification languages take over the definitions used in the corresponding modeling and programming languages without investigating potential inconsistencies between formal and object-oriented concepts. For example, the referential semantics of object-orientation and the dynamical binding of operations competes with the value semantics of most formal approaches. The possibility of extending specifications incrementally via inheritance and polymorphism leads to ambiguous specifications. The imperative state changes of the objects cannot be expressed by functional specifications because they do not allow side effects. Object-oriented concepts are often only used for internal structuring within existing approaches to a formal specification, e.g. object-oriented extensions to Z like Object-Z or OOZE ([LaHa1994b]). Formal approaches based on the object-oriented computation model are rarely found.

Today formal approaches are seldom used for practical purposes although they play an increasingly important role in the development of large class libraries and in improving reuse.

Concurrency and Distribution (4)

The fourth factor relates to the trend towards distributed and concurrent applications. Object orientation could prove to be a good basis for such applications. The metaphors „objects as agents“ ([Nier1992]), „active objects“ ([Nier1993]) and „objects as machines“ ([Meye1997], pp. 751 ff.) suggest the development of loosely coupled program parts operating independently and at the same time. According to this view object systems are message-coupled „multi-processor systems“. Therefore it comes as no surprise that a number of parallel object-oriented programming languages exists. But these languages involve several unsolved problems such as inheritance anomaly or typing problems in the context of polymorphism ([MaYo1993]) so that conventional sequential languages, possibly extended by the simple parallel concept *process* or *thread*, still prevail. In these languages the concepts process and object are orthogonal, i.e. independent from each other. But with the number of agents and hence of potential communication channels the amount of synchronization increases, too. In the case of sequential languages this requires greater effort during software development because no explicit synchronization constructs exist. For this reason synchronization and communication should be part of the concept „object“.

Hence, although currently of minor importance, strongly coupling concurrency and object orientation seems to be a vital prerequisite for developing distributed applications in the context of global networks object-orientedly.

Persistence and Multi-User Systems (5)

The fifth factor is determined by the application domain. Almost any system of considerable size (e.g. a corporate information system) has to store large amounts of data and is accessed by multiple users. Especially in modeling, a number of additional requirements has to be taken into account for multi-user systems with persistent data, e.g. deciding which objects (data) are persistent, which objects can be accessed by more than one user and how access to and visibility of objects and their methods can be managed. Up to now support of the modeling approaches for persistent objects is often restricted to database support, mapping objects to tables and links to relational databases ([BIPr1998]).

Object-oriented databases offer a proved technology for the integration of persistence, multi-user features and object orientation. But unfortunately the semantics of object-oriented concepts of databases differs from that of modeling languages as well as from that of programming languages. For example: do members of a subclass also belong to the extent (definition) of the corresponding superclass

([LaVo1997])? Is it possible to inherit from objects? The diversity of database languages themselves presents another problem. It is therefore an interesting case for research if programming and database language coincide like e.g. in Gemstone with Smalltalk. Although this reduces the language independence of the database, it allows a „normal“ object-oriented development with a persistent multi-user Smalltalk ([Gems1996], [Alma1995]). Moreover the message passing of object-oriented languages enables queries providing at least the expressive power of typical query languages such as SQL.

To achieve a seamless software engineering modeling languages should support concepts for treating access control and persistence of objects. We can thereby deploy the whole potential of object orientation for the modeling of information systems. Furthermore it should be possible to develop database and application system from the same object-oriented viewpoint so that a costly mapping to persistent structures (e.g. tables) becomes obsolete. So object and database experts have to work together more closely.

Components (6)

The last factor describes a topic which is intensively discussed at the moment in application software engineering, namely components ([Szyp1997]). Software is to be put together from prefabricated components in the manner of a kit. The components offer their functionality as a service via well-defined interfaces. They are composed according to standardized rules with the aim of developing modular systems from prefabricated components which are easily interchangeable. Hence component-based software engineering is strongly based on the idea of independent and easily composable modules. It can be seen as a realization of the software-IC metaphor (cf. [Cox1986]). But regarding the term component itself there is no unanimity concerning its definition. Some of the definitions are even partly contradictory ([Szyp1997]). Moreover the relation between object orientation and component-based software remains an open issue. The views on this relation range from “components *are* objects” to “components are a bounded piece of software without a state but with well-defined interfaces (e.g. a set of classes)” (cf. [Szyp1997]). Many of the ideas such as frameworks and design patterns currently discussed in the context of components have object-oriented roots.

Nevertheless object orientation can profit from results of component research where for the first time industrial-scale reuse, protection of ownership rights, developing a component market and economic aspects such as marketing are studied and, even more important, put to practice (at least in parts). Beyond this, technical mechanisms such as frameworks, design patterns and meta programming, which are useful for the composition, integration, configuration and flexible adaptation of components, can also be adapted well to objects in many cases. A further enrichment of object orientation comes from structuring systems with the help of coarse-grain object structures or design patterns such as facet for delineating subsystems ([GHJV1995]).

Challenges and Chances of Integration

Up to this point we have investigated each factor separately showing the potential and the problems of a bilateral integration with object orientation. But a more challenging and at the same time more rewarding aim is the consideration of all aspects in a unified framework. Each factor highlights only a single facet of the modeled system. But in reality these facets are highly interdependent and a singular treatment amounts to “cutting” these dependencies. Hence an integrated view offers considerable advantages. Still the realization of these chances requires considerable effort in overcoming major obstacles on the way. For example, the combination of persistence and distributed objects causes the same problems known from distributed databases (cf. [OzVa1998]).

Conclusion

Currently we can observe a fast development of scientific knowledge in almost all areas of computer science including object-oriented software engineering. Hence one might argue whether computer science and the object community have already reached the mature state of a discipline called “normal science” by Kuhn (see [Kuhn1962]). The integration of the aforementioned factors represents a major mi-

lestone towards such an established scientific field. The important driving forces behind this development are the increasing requirements posed by applications and their users.

The aim of this paper is to plead for an integration of vital success factors with the object-oriented core. We take a conservative point of view arguing that each new development should be seen in the light of how much it contributes to the existing object-oriented techniques. In the literature each factor is typically investigated separately whereas we suggest to bring these fields together to better fulfil the combined requirements of future software.

References

- [AbCa1996] Abadi, M.; Cardelli, L.: *"A Theory of Objects"*. Springer. New York. 1996.
- [AWY1993] Agha, G.; Wegner, P.; Yonezawa, A.: *"Research Directions in Concurrent Object-Oriented Programming"*. MIT Press. Cambridge, Massachusetts. 1993.
- [Alma1995] Almarode, J.: *"Multi-User Smalltalk"* in The Smalltalk Report. vol. 4. no. 4. pp. 27-30. January 1995.
- [BIPr1998] Blaha, M.; Premerlani, W.: *"Object-Oriented Modeling and Design for Database Applications"*. Prentice Hall. New York. 1998.
- [Booc1994] Booch, G.: *"Object-Oriented Design with Applications"*. Benjamin Cummings. Readwood City. 1994.
- [BCC+1995] Bruce, Kim B.; Cardelli, Luca; Castagna, Guiseppe; Eifrig, Jonathan; Leavens, Gary T.; Pierce, Benjamin; Smith, Scott; Trifonov, Valery *"On Binary Methods"* in Theory and Practice of Object Systems. vol. 1. no. 3. pp. 221-242. 1995.
- [CaWe1985] Cardelli, L.; Wegner, P.: *"On Understanding Types, Data Abstraction, and Polymorphism"*. in Computing Surveys. vol. 17. pp. 471-522. Dezember 1985.
- [Cox1986] Cox, B.: *"Object Oriented Programming: An Evolutionary Approach"* Addison Wesley. Reading, Massachusetts. 1986.
- [DMN1970] Dahl, O.-J.; Myhrhaug, B.; Nygaard, K.: *"Simula 67 common base language"* Report N.S-22. Norwegian Computing Center. Oslo. 1970.
- [FHGP1996] Firesmith, D.; Henderson-Sellers, B.; Graham, I.; Page-Jones, M.: *"OPEN Modeling Language (OML). Reference Manual"*. Version 1.0. 8 December 1996. (www.csse.swin.edu.au/OPEN/comn.html)
- [Gems1996] *"GemStone Documentation"* Version 5.0. GemStone Systems, Inc. July 1996.
- [Gold1998] Goldberg, Adele: *"The Community of Smalltalk"* in [Salu1998] pp. 51-94.
- [GHJV1995] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *"Design Patterns. Elements of Reusable Object-Oriented Software"*. Addison-Wesley. Reading, Massachusetts. 1995.
- [HaOs1993] Harrison, W.; Ossher, H.: *"Subject-Oriented Programming (A Critique of Pure Objects)"*, in Proceedings of OOPSLA '93. pp. 411-428. 1993.

- [HGY1997] Henderson-Sellers, B.; Graham, J.; Younessi, H.: *"The OPEN Process Specification"* Addison-Wesley. Harlow. 1997.
- [JBR1999] Jacobson, I.; Booch, G.; Rumbaugh, J.: *"The Unified Software Development Process"*. Addison-Wesley. Reading, Massachusetts. 1999.
- [Kay1993] Kay, A.: *"The Early History of Smalltalk"*. in ACM SIGPLAN Notices. vol. 28. no. 3. pp. 69-95. March 1993.
- [KLM+1997] Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C. V.; Loingtier, J.-M.; Irwin, J.: *"Aspect-Oriented Programming"* in Proceedings of ECOOP '97. pp. 220-242. LNCS 1241. Springer. New York. 1997.
- [Kuhn1962] Kuhn, T. S.: *"The Structure of Scientific Revolutions"*. University of Chicago. 1962.
- [LaHa1994a] Lano, K.; Haughton, H.: *"A Comparative Description of Object-Oriented Specification Languages"*. in [LaHa1994b]. pp. 20-54.
- [LaHa1994b] Lano, K.; Haughton, H.: *"Object-Oriented Specification Case Studies"*. Prentice-Hall. New York. 1994.
- [LaVo1997] Lausen, G.; Vossen G.: *"Models and Languages of Object-Oriented Databases"*. Addison-Wesley. Reading, Massachusetts. 1997.
- [LRP+1995] Lewis, T.; Rosenstein, L.; Pree, W.; Gamma, E.; Calder, P.; Andert, G.; Vlassides, J.; Schmucker, K.: *"Object Oriented Application Frameworks"*. Manning. Greenwich. 1995.
- [Lieb1996] Lieberherr, K.: *"Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns"*. PWS Publishing Company. Boston. 1996.
- [MaYo1993] Matsuoka, S.; Yonezawa, A.: *"Analysis of Inheritance Anomaly in Object-Oriented Concurrent Programming Languages"* in [AWY1993] pp. 107-150.
- [Mey1997] Meyer, B.: *"Object-oriented Software Construction"*. Prentice Hall. New York. 1997.
- [Nier1992] Nierstraß, O.: *"Towards an Object Calculus"*. in Proceedings of the ECOOP '91 Workshop on Object-Based Concurrent Computing. LNCS 612. Springer. New York. pp. 1-20. 1992.
- [Nier1993] Nierstraß, O.: *"Composing Active Objects"*. in [AWY1993] pp. 151-171.
- [OMG1999] *"OMG Unified Modeling Language Specification"*. Version 1.3 alpha R2. Object Management Group. Framingham Mass. January 1999. (www.omg.org)
- [OzVa1998] Ozsu, M. T.; Valduriez, P.: *"Principles of Distributed Database Systems"*. Prentice Hall. Englewood Cliffs. 1998.
- [Parn1972] Parnas, D. L.: *"On the Criteria To Be Used in Decomposing Systems Into Modules"* in Communications of the ACM. vol. 15. no. 12. pp. 1053-1058. December 1972.
- [Pres1997] Pressman, R. S.: *"Software Engineering: A Practitioner's Approach"*. McGraw-Hill. New York. 1997.

New York. 1997.

- [RBPL1991] Rumbaugh, J. ; Blaha, M.; Premerlani, F. E.; Lorensen, W.: "*Object-Oriented Modeling and Design*". Prentice Hall. New York. 1991.
- [Salu1998] Salus, P. H.: "*Handbook of Programming Languages: Object-Oriented Programming Languages*". Vol. 1. Macmillan. Indianapolis. 1998.
- [Szyp1997] Szypersky, C.: "*Component Software: Beyond Object-Oriented Programming*". Addison-Wesley. Reading, Massachusetts. 1997.
- [TsNi1989] Tsichritzis, D.; Nierstraß, O.: "*Directions in Object-Oriented Research*" in Kim, W.; Lochovsky F. H.: *Object-Oriented Concepts, Databases and Applications*. pp. 523-536. ACM Press. New York. 1989.