Simula – von der Simulation zur ersten objektorientierten Programmiersprache

Jürgen Nickelsen
Stephan Rudlof

Referat in der Lehrveranstalung
“Objektorientierte Programmierung und Modellierung”
am 18.4.1995

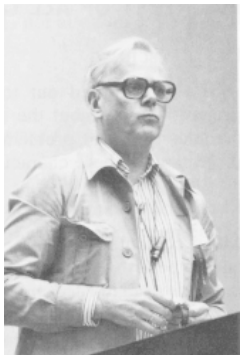
Die Sprache Simula gilt als erste objektorientierte Programmiersprache. In dieser Ausarbeitung erzählen wir von ihrer Entstehungsgeschichte, schildern einige ihrer Eigenschaften und zeigen ihren Einfluß auf die Disziplin der objektorientierten Programmierung.

1.0 Historischer Zusammenhang

Simula wurde in den sechziger Jahren am Norwegian Computing Center (NCC) von Kristen Nygaard und Ole-Johan Dahl entwickelt. Hintergrund waren ihre Arbeiten im Bereich Operations Research am Norwegian Defense Research Establishment (NDRE), einer Forschungseinrichtung des Norwegischen Militärs. Am NCC sollte eine Sprache zur Simulation zeitdiskreter Systeme entwickelt werden.

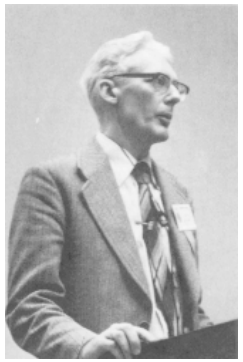
1.1 Die Autoren

Kristen Nygaard



Kristen Nygaard arbeitete seit 1948 am NDRE, wo er seinen Militärdienst abgeleistet hatte. 1956 schrieb er seine Diplomarbeit über die Anwendung von Monte-Carlo-Methoden in der Simulation. 1960 ging Kristen Nygaard zum NCC. Ab 1976 war er Professor in Aarhus (Dänemark).

Ole-Johan Dahl



Ole-Johan Dahl fing 1952 an, am NDRE zu arbeiten. 1957 schrieb er seine Diplomarbeit über “Numerical Mathematics”. 1963 ging Ole-Johan Dahl zum NCC. Ab 1968 arbeitete er als Professor in Oslo.

1.2 Zeitliche Einordnung

	Simula	Rest der Welt
1957		FORTRAN III
1960		Algol 60, LISP, COBOL
1961	erste Ideen Simula I	
1963	Sprachdefinition	
1964	erster Compiler	
1966		Algol W (Wirth)
1968	Simula 67	Algol 68 Report (erste Fassung)
1969		Pascal
1971	Installation in Karlsruhe	
1972	Compiler für IBM 370	

Als die Entwicklung von Simula I begann, waren die dominierenden Programmiersprachen der sechziger Jahre, FORTRAN, Algol 60, LISP und COBOL, schon bekannt.

2.0 Der Weg zu Simula I

Die Entwicklung von Simula wurde mit dem Ziel begonnen, ein Tool zur Simulation zeitdiskreter Systeme zu entwickeln. Die wesentlichen Ideen waren dabei:

- Zwischen dem Programmtext und seiner Ausführung muß unterschieden werden. Zur Laufzeit werden dynamische Instanzen von Entitäten erzeugt.
- Daten und die Operationen auf diesen Daten gehören zusammen.
- Die zu simulierenden Systeme werden durch eine Anzahl von Prozessen modelliert.

2.1 Entwicklungsprozeß

Die Ideen, die zur Entwicklung von Simula geführt haben, sind in sehr hohem Maße in der Interaktion zwischen Kristen Nygaard und Ole-Johan Dahl entstanden. Diese sind deshalb — wie beide in [Nygaard, Dahl 1981] selber bemerken — nicht jeweils einem einzelnen von beiden zuzuschreiben. Dahl war vorher im wesentlichen als Sprachenentwickler und Programmierer, Nygaard im Operations Research tätig.

Die ersten Ideen für eine Sprache, die zur Systembeschreibung und zur Simulation geeignet sein sollte, kamen vom NCC. Nygaard war

dort seit 1960 tätig. Die erste schriftliche Referenz auf Simula gab es am 5.1.1962 von Nygaard [Nygaard, Dahl 1981].

An dieser Stelle wurde auch schon ein „Expert Programmer“ erwähnt: Damit war Dahl gemeint, zu dem Nygaard im Dezember 1961 Kontakt aufnahm, nachdem die ersten, scheinbar mächtigen Ideen aufgetaucht waren [Dahl, Nygaard 1981]. Im Mai 1962 hatten die beiden den Eindruck, einen Sprachenvorschlag zu haben, den sie anderen Leuten präsentieren könnten. Mitte März 1963 wechselte Dahl zum NCC, um sich voll der Arbeit an Simula widmen zu können.

Beim NCC gab es zunächst wenig Begeisterung für eine solche Sprache. Es wurde behauptet [Nygaard, Dahl 1981]:

- Sie sei nutzlos;
- sie sei nützlich, aber sie wurde schon vorher entwickelt;
- die Ideen seien nicht gut genug;
- Nygaard und Dahl würde die Kompetenz für solch ein Projekt fehlen, und es würde deshalb niemals fertig werden;
- Norwegen sei ein zu kleines Land für solch ein großes Projekt.

Die Entwicklung von Simula konnte trotzdem fortgeführt werden, da die Leitung des NCC das Projekt unterstützte.

2.2 Wichtige Entwicklungsphasen von Simula I

1. Von Sommer 1961 bis zum Herbst 1962 gab es erste Ideen, die auf dem mathematischen „Discrete Event Network“-Konzept und allgemeinen Sprachüberlegungen (ohne konkrete Implementierung) basierten.
2. Vom Herbst 1962 bis zum September 1963 wurde die Idee verfolgt, einen Präprozessor für ALGOL 60 mit einem „Procedure Package“ zu schreiben, um Simula zu realisieren.
3. Die Idee mit dem Präprozessor wurde verworfen zugunsten der Entscheidung, einen guten (schon vorhandenen) ALGOL-60-Compiler zu modifizieren und zu erweitern; dabei sollte eine neue Speicherverwaltung verwendet werden, die von Dahl entwickelt worden war.
4. Die Implementierung des Simula-I-Compilers fand von März bis Dezember 1964 statt; die Hauptreferenz zu dem fertigen Produkt Simula I ist [Dahl, Nygaard 1965].

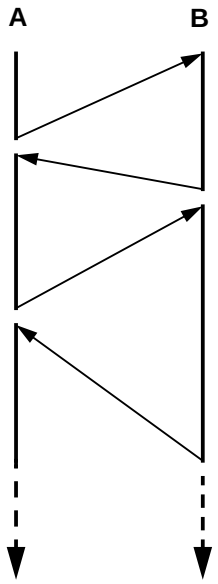


Bild 1. Kontrollfluß bei Koroutinen in Simula

2.3 Grundlegende Spracheigenschaften von Simula I

- Die Sprache basiert auf ALGOL 60.
- Es gibt dynamisch instanziierbare Koroutinen, auch Prozesse genannt.
- Koroutinen enthalten lokale Prozeduren und Daten.
- Zugriff auf Daten anderer Koroutinen ist möglich.
- Zur Verwaltung von Koroutinen werden Mengen bzw. Listen benutzt.
- Es gibt eine automatische Speicherverwaltung, die einen Reference Count Garbage Collector benutzt.

2.3.1 Koroutinen.

Koroutinen sind quasi-parallel ablaufende Teile des Programms. Sie können dynamisch erzeugt werden. Der Wechsel zwischen ihnen erfolgt nicht zeitscheiben- oder ereignisgesteuert (wie zum Beispiel zwischen Prozessen bei UNIX), sondern explizit über entsprechende

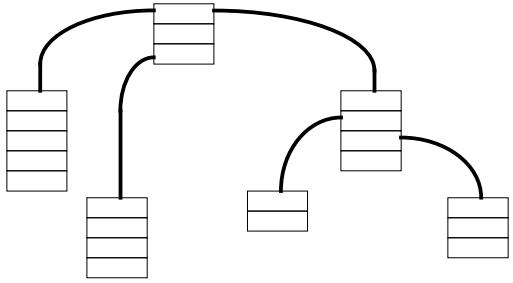


Bild 2. Multiple Stacks in Simula

Konstrukte der Sprache. Es läuft immer nur eine Koroutine gleichzeitig, obwohl mehrere gleichzeitig aktiv sein können. Ein sehr ähnliches Koroutinenkonzept finden wir später bei Modula-2.

In Bild 1 sehen wir symbolisch den Kontrollfluß zweier Prozeduren A und B, die als Koroutinen laufen. A wird zuerst aufgerufen und gibt die Kontrolle an B ab. Daraufhin beginnt B zu laufen. Wenn nun eine Koroutine die Kontrolle an die andere abgibt, läuft diese an der Stelle weiter, an der sie vorher die Kontrolle an die andere abgegeben hat.

2.3.2 Speicherkonzept

Bei ALGOL 60 wurde nur ein Stack benötigt, da eine aktive Prozedur erst dann weiterlaufen konnte, wenn von ihr aufgerufene Prozeduren terminiert waren. Diese strikte Sequentialität gibt es in Simula nicht mehr. Da eine Koroutine auch schon weiterlaufen kann, bevor die von ihr aufgerufenen anderen Koroutinen terminiert sind, muß sie unabhängig von diesen auf ihre lokalen Daten zugreifen können. Dieses Problem wurde von Dahl und Nygaard dadurch gelöst, daß jede Koroutine ihren eigenen Stack bekam (siehe Bild 2). So konnte der Zugriff auf lokale Daten der jeweiligen Koroutine unabhängig von den anderen gemacht werden.

3.0 Simula 67

3.1 Ziele

Die Erfahrungen mit Simula I bewogen Dahl und Nygaard, die Sprache mit folgenden Vorgaben weiterzuentwickeln:

- Algol 60 sollte weiterhin als Basissprache dienen.
- Die Implementierung sollte verbessert werden. Gerade bei der Speicherverwaltung wies Simula I deutliche Defizite auf; der Garbage Collector war zu langsam.
- Die Möglichkeiten zur Systemsimulation sollten verbessert werden.
- Die neue Sprache sollte eine allgemein verwendbare Programmiersprache sein.
- Die Wiederverwendbarkeit von Prozeßbeschreibungen sollte verbessert werden.
- Die Sprache sollte einen „hohen Standard“ erreichen. Das beinhaltete unter anderem:
  - Die Compiler und der von ihnen generierte Code sollten so schnell sein wie bei den besten verfügbaren Algol-60-Compilern.
  - Gute Dokumentation und Lehrbücher sollten verfügbar gemacht werden.
- Die Sprache sollte eine „existierende Sprache“ werden. Das hieß:
  - Sie sollte auf allen wichtigen Plattformen verfügbar sein.
  - Sie sollte eine große Verbreitung erreichen.
  - Sie sollte Einfluß auf zukünftige Entwicklungen bei Programmiersprachen nehmen.

3.2 Eigenschaften der Sprache

Die Sprache Simula 67 ist weitgehend kompatibel zu Algol 60. Damit ist auch sie eine „klassische“ imperative Programmiersprache mit einem Hauptprogramm, Blockstruktur und explizitem Kontrollfluß. Durch folgende Punkte unterscheidet sie sich von anderen Sprachen dieser Art:

- Alle Variablen werden automatisch initialisiert. Dabei wird jeweils ein vom Typ der Variablen bestimmter Standardwert benutzt; bei numerischen Typen ist das zum Beispiel die Null.
- Simula hat ein Klassenkonzept, das im wesentlichen heutigen objektorientierten Sprachen sehr ähnelt. (Siehe dazu Abschnitt 3.2.1., „Simulas Klassenkonzept“.)
- Prozesse werden duch Koroutinen modelliert, die als Klassenexemplare implementiert sind. Sie können damit beliebig dynamisch erzeugt werden. Es gibt Sprachmittel zur Weitergabe des Kontrollflusses zwischen Koroutinen.

- Die Klassen `SIMSET` und `SIMULATION` bieten Unterstützung für zeitdiskrete ereignisorientierte Simulation. Besonders wichtig ist die Anweisung `activate` *prozeß klausel* bzw. `reactivate` *prozeß klausel*. Dabei ist *prozeß* eine Koroutine; *klausel* eine der folgenden Klauseln:
  - `at` *zeitpunkt*
    Aktiviere *prozeß* zur Zeit *zeitpunkt*.
  - `delay` *zeitraum*
    Aktiviere *prozeß* nach der Zeit *zeitraum*.
  - `before` *anderer_prozeß*
    Aktiviere *prozeß* vor *anderer_prozeß*.
  - `after` *anderer_prozeß*
    Aktiviere *prozeß* nach *anderer_prozeß*.

  Bei diesen Primitiven muß man sich vor Augen halten, daß es hier um Simulation geht, nicht um Echtzeitverarbeitung. Die Zeit, von der hier die Rede ist, ist nicht die reale Zeit, sondern die virtuelle Zeit im simulierten Modell.
- Auch Simula 67 bietet automatische Speicherverwaltung. Um die Performance bei größeren Speicheranforderungen zu verbessern, wurde ein aus der LISP-Community übernommener kompaktierender Garbage Collector implementiert.
- Eine nennenswerte Bibliothek an Standardklassen bietet Simula 67 nicht. Einzig doppelt verkettete Listen wurden implementiert, die Dahl und Nygaard für die Simulation für besonders nützlich hielten.

3.2.1 Simulas Klassenkonzept

Simula 67 enthält das Konzept von Klassen von Objekten. Für eine Klasse können Attribute (Variablen) und Methoden (Routinen) definiert werden. Für Attribute können explizite Initialisierungen definiert werden, die von der automatischen Initialisierung abweichen. Klassenexemplare (Objekte) werden explizit erzeugt. Bei der Erzeugung werden die für die Attribute definierten Initialisierungen ausgeführt.

Eigenschaften einer Klasse können von einer anderen Klasse geerbt werden; die Wiederverwendbarkeit von Code wird dadurch gefördert. Eine Klasse kann nur die Eigenschaften einer einzigen Klasse erben; Mehrfachvererbung gibt es nicht.

In einer Klasse können „virtuelle“ Methoden definiert werden, die erst in den Erben dieser Klasse implementiert werden. Dadurch können polymorphe Funktionen und Prozeduren realisiert werden.

Eine Klasse kann innerhalb einer anderen Klasse definiert werden. In diesem Fall ist die so definierte Klasse nur lokal innerhalb der umschließenden Klasse bekannt.

Der Sprachstandard Simula 67 erlaubte noch den unbeschränkten Zugriff auf alle Attribute eines Objekts. In einem späteren Standard wurden Möglichkeiten eingeführt, diesen Zugriff im Sinne des Information Hiding einzuschränken. Mit der Markierung „protected“ versehene Attribute können nur von dem Objekt selbst aus erreicht werden; mit der Markierung „hidden“ versehene sogar nur dann, wenn das Objekt der Klasse angehört, die dieses Attribut definiert hat, also nicht von Objekten abgeleiteter Klassen aus.

4.0 Einfluß und Weiterentwicklungen

Wie wir heute wissen, hat die objektorientierte Vorgehensweise, die in Simula zum ersten Mal in einer Programmiersprache verwirklicht wurde, einen erheblichen Einfluß auf die weitere Entwicklung von Programmiersprachen gehabt. Die Sprachen Smalltalk, C++ und Beta bauen direkt auf den mit Simula gewonnen Erfahrungen auf, andere, zum Beispiel Eiffel, die wiederum davon beeinflußt wurden, entwikkelten das Konzept der objektorientierten Programmierung weiter.

Alan Kay schreibt in [Kay 1993], daß ihm das Konzept von strukturierten Daten, denen direkt dazugehörige Operationen zugeordnet sind, schon sehr früh begegnet ist. In dem objektorientierten interaktiven Zeichenprogramm „Sketchpad“ wurde dieses Konzept auf Vererbung ausgedehnt. Sehr eindrücklich beschreibt er, wie er dieses Konzept in Simula wiederfindet und auf der Basis dieser Ideen beginnt, Smalltalk zu entwickeln.

Bjarne Stroustrup erzählt in [Stroustrup 1993] von den Erfahrungen, die er in den frühen siebziger Jahren mit Simula als Simulationssprache macht. Er ist beeindruckt von der Eleganz, mit der Simula mit seinen Klassen und Koroutinen die Modellierung von Systemen unterstützt. Da die Systemumgebung von der Performance her sehr unbefriedigend ist, reimplementiert er sein System in BCPL, hat dann aber mit dem unzureichenden Typsystem und der mangelnden Unterstützung durch die Sprache im allgemeinen zu kämpfen. Aufbauend auf diesen Erfahrungen beginnt er, nach dem Vorbild Simulas eine objektorientierte Erweiterung von C, „C with Classes“, zu entwikkeln, aus der später C++ entsteht.

Kristen Nygaard entwickelte später mit anderen die Programmiersprache Beta, in der Konzepte objektorientierter und funktionaler Programmierung durch sogenannte „Patterns“ verallgemeinert werden. Persistenz von Daten und ausgefeilte Konzepte von Nebenläufigkeit und Prozeßkommunikation sind weitere Schwerpunkte dieser Sprache. [Knudsen 1995]

Kaum eine Programmiersprache ist davon „verschont“ geblieben, um objektorientierte Konzepte erweitert zu werden, darunter Pascal, Forth, COBOL und Ada. Common Lisp ist mit CLOS (Common Lisp Object System) zu einer nicht unbedeutenden Plattform für objektorientierte Software-Entwicklung geworden. Sogar die funktionale Programmierung ist davon beeinflußt worden: Haskells erweiterbares hierarchisches Typsystem lehnt sich bewußt daran an, obwohl es auf Objekte mit einem veränderbaren Status verzichtet.

5.0 Simula heute

Simula (die „67“ wurde mittlerweile aus dem Namen gestrichen) ist heute noch, wie Dahl und Nygaard es beabsichtigt hatten, eine „existierende Sprache“. Sie hat „eine zwar kleine, aber leidenschaftliche Fangemeinde“ [Meyer 1990]. Eine Standardgruppe sorgt für die Pflege der Sprachdefinition; der letzte Standard datiert von 1986. Die „Association of Simula Users“ (ASU) vertritt die Interessen der Simula-Anwender und koordiniert deren Aktivitäten.

Für verschiedene Plattformen sind kommerzielle Compiler-Implementierungen verfügbar, hauptsächlich von Firmen aus Norwegen und Schweden. Für Unix-artige Systeme gibt es einen frei verfügbaren Compiler (CIM) mit einem portablen C-Backend [Johansen, Mjoes, Krogdahl 1995]. Die zur Zeit aktuelle Version 1.63 ist von Ende 1994. Es gelang uns mit diesem Compiler, ein Beispielprogramm aus [Birtwistle et al. 1973] zu kompilieren und laufen zu lassen (siehe Abschnitt 6, „Ein Beispielprogramm“).

Es existiert eine BITNET-Mailingliste für Simula. Diese Mailingliste wird ins USENET gespiegelt; in der entsprechenden Newsgroup finden sich einige Artikel aus den letzten paar Wochen.

Für den Editor Emacs gibt es eine spezielle Anpassung zum Schreiben von Simula-Programmen [Eriksen 1992]. Die in der aktuellen Distribution von XEmacs (19.11) enthaltene Version ist von 1992.

6.0 Ein Beispielprogramm

Das folgende Beispielprogramm haben wir [Birtwistle et al. 1975] entnommen und mit dem Compiler CIM kompiliert.

```
SIMSET
BEGIN

   LINK CLASS SPIELER(IDENT); INTEGER IDENT;
   BEGIN INTEGER PUNKTE;
      DETACH;
      WHILE TRUE DO
      BEGIN PUNKTE := PUNKTE + WURF.AUGEN;
         OUTINT(IDENT,7); OUTINT(PUNKTE,8); OUTIMAGE;
         IF PUNKTE < 30 THEN
           RESUME(IF SUC=/=NONE THEN SUC ELSE SPIEL.FIRST)
         ELSE
           DETACH
      END
    END SPIELER;

    HEAD CLASS RUNDE(ANZAHL); INTEGER ANZAHL;
    BEGIN INTEGER NUMMER;
       FOR NUMMER := 1 STEP 1 UNTIL ANZAHL DO
          NEW SPIELER(NUMMER).INTO(THIS RUNDE);
       OUTTEXT("SPIELER  PUNKTE "); OUTIMAGE;
    END RUNDE;

    CLASS WUERFEL;
    BEGIN INTEGER PROCEDURE AUGEN; AUGEN:=RANDINT(1,6,HOLD);
       INTEGER HOLD;
       WHILE HOLD < 1 OR MOD(HOLD,2)=0 DO
       BEGIN OUTTEXT("EINGABE EINER POSITIVEN UNGERADEN ZAHL:");
          OUTIMAGE; OUTIMAGE;
          HOLD := ININT
       END
    END WUERFEL;


    REF(WUERFEL)WURF; REF(RUNDE)SPIEL;
    WURF :- NEW WUERFEL;
    SPIEL :- NEW RUNDE(3);
    RESUME(SPIEL.FIRST)

END
```


Hier ein Protokoll des Compiler- und Programmlaufs:

```
$ cim wuerfel.sim
Compiling wuerfel.sim:
cc -O -c wuerfel.c:
cc  -o wuerfel wuerfel.o -L/usr/local/lib/cim /usr/local/lib/
cim/libcim.a  -lm :
$ wuerfel
EINGABE EINER POSITIVEN UNGERADEN ZAHL:

45
SPIELER  PUNKTE
      1       5
      2       5
      3       1
      1       8
      2       7
      3       7
      1      10
      2       9
      3      13
      1      15
      2      14
      3      18
      1      21
      2      20
      3      21
      1      23
      2      26
      3      25
      1      26
      2      28
      3      27
      1      32
```

7.0 Zusammenfassung

Die objektorientierte Programmiersprache Simula wurde Ende der sechziger Jahre in Norwegen von Kristen Nygaard und Ole-Johan Dahl während ihrer Arbeit am Norwegian Computer Center (NCC) entwickelt. Grundlage dafür waren Algol 60 und die vorher von ihnen entwickelte reine Simulationssprache Simula I, die ebenfalls auf Algol 60 basierte.

Simula enthält ein Klassenkonzept, bei dem die Klassendefinition Attribute, Methoden und die Objektinitialisierung enthält. Klassenexemplare (Objekte) können während des Programmablaufs dynamisch erzeugt werden. Vererbung von Klasseneigenschaften wird unterstützt, aber keine Mehrfachvererbung. Information Hiding wurde in einem späteren Sprachstandard hinzugefügt. Die Speicherverwaltung wird durch Garbage Collection unterstützt. Für die Simulation zeitdiskreter ereignisorientierter Systeme bietet Simula ein Koroutinenkonzept und spezielle Primitiven zur Prozeßsynchronisation.

Simula hatte mit dem Konzept der objektorientierten Programmierung erheblichen Einfluß auf die weitere Entwicklung von Programmiersprachen. In vielen Sprachen ist heute ein von Simula abgeleitetes Klassenkonzept zu finden.

Heute ist Simula nicht sehr weit verbreitet, wird aber noch aktiv unterstützt. Es gibt Compiler für verschiedene Plattformen, darunter einen frei verfügbaren.

8.0 Literatur

G. M. Birtwistle et al.: *SIMULA BEGIN*. Studentlitteratur, Schweden, und Auerbach Publ. Inc., Philadelphia, Pa., 1973.

Ole-Johan Dahl, Kristen Nygaard: *SIMULA — A language for programming and description of discrete event systems. Introduction and user's manual*. NCC Publ. No. 11. (D), 1965.

Hans Henrik Eriksen: *simula.el — SIMULA 87 code editing commands for Emacs*, 1992. In *XEmacs 19.11*. Software-Distribution, University of Illinois, September 1994.

Jan Rune Holmevik: *Compiling Simula*. In *Annals of the History of Computing*. Vol. 16(4), 1994.

Sverre Hvammen Johansen, Terje Mjoes, Stein Krogdahl: *Cim — Simula Compiler based on the C programming language*. Software-Distribution Version 1.63, Department of Informatics, University of Oslo, Februar 1995.

Alan Kay: *The Early History of Smalltalk*. In *History of Programming Language Conference (HOPL-II) Preprints*, ACM SIGPLAN Notices, Volume 28, No 3, März 1993.

Jorgen Lindskov Knudsen (ed.): *FAQ: BETA Programming Language*. In *comp.lang.beta*, *comp.answers*, *news.answers* (USENET), <3km3t6$rvu@belfort.daimi.aau.dk>, Mai 1995.

C. H. Lindsey: *A History of Algol 68*. In *History of Programming Language Conference (HOPL-II) Preprints*, ACM SIGPLAN Notices,
Volume 28, No 3, März 1993.

Bertrand Meyer: *Objektorientierte Softwareentwicklung*. Hanser, Wien 1990.

Kristen Nygaard, Ole-Johan Dahl: *The Development Of The Simula Language*. In *History of Programming Languages*; Academic Press, 1981.

Bjarne Stroustrup: *A History of C++*. In *History of Programming Language Conference (HOPL-II) Preprints*, ACM SIGPLAN Notices, Volume 28, No 3, März 1993.

USENET-Newsgroup *bit.listserv.simula*, Juni 1995.

Niklaus Wirth: *Recollections of the Development of Pascal*. In *History of Programming Language Conference (HOPL-II) Preprints*, ACM SIGPLAN Notices, Volume 28, No 3, März 1993.