Thesen zur Programmierung

Prof. Dr.rer.nat. Herbert Klaeren
Universität Tübingen

3. April 1995

1. Eine der Lehren aus der Software-Krise ist es, daß Software nicht nach dem Bastler-Verfahren hergestellt werden darf, sondern solider Grundlagen bedarf. (Das englische Wort *hacker* läßt sich in Bedeutung und Konnotation mit dem deutschen *Fummler* gleichsetzen.)

2. Solide Grundlagen für die Programmierung können nur entstehen, wenn vor dem Programmieren sauber und formal spezifiziert wird, was die Leistung des Programms sein soll [10, Kap. 2].

3. Für die Grundlagen-Ausbildung in der Programmierung genügt es, sich auf *funktionale Spezifikationen* zu beschränken; diese beschreiben die vom Programm zu erbringende Funktion und lassen Leistungsdaten, Benutzerschnittstellen, Projektorganisation und andere für die Praxis wichtige Details zunächst außer acht.

4. Funktionale Spezifikationen lassen sich ausgezeichnet in der Begriffswelt der *Algebraischen Spezifikation* [2, 3] formulieren. Hier ist ein weites Spektrum möglich, von nicht-konstruktiven Gleichungsspezifikationen bis hin zu konstruktiven, Programm-ähnlichen Formalismen.

5. Besonders wichtig ist es, frühzeitig auf das *Abstraktionsprinzip* („information hiding“) [10, Kap. 6] hinzuerziehen. Gerade im Kontext der abstrakten Datentypen ist die algebraische Spezifikationstechnik aufgekommen.

6. Die neunziger Jahre sind die Dekade der *Objektorientierung*; diesem Thema muß deshalb bei der Programmierausbildung Gewicht geschenkt werden. Allerdings haben die „Objekte“ der jetzt so modernen objektorientierten Programmier-Ideen eine große Verwandtschaft zu den abstrakten Datentypen [19]; in Anlehnung an J.A. Goguen könnte man Objekte als Datentypen mit einem inneren Zustand beschreiben. In der Tat ist das am häufigsten verwendete Beispiel für einen abstrakten Datentyp, der Kellerspeicher, in Wirklichkeit ein Objekt im Sinne der objektorientierten Programmierung.

7. Ein grundlegendes Verständnis der Prinzipien der Softwaretechnik („Software Engineering“) ist auch dann wichtig, wenn man glaubt, nur die „Programmierung im Kleinen“ betreiben zu wollen.

8. Am Software-Lebenslauf gemessen, gehören algebraische Spezifikationen in den Bereich zwischen Entwurfsspezifikation („Grobentwurf“) und Programmierung.

9. Es ist unrealistisch, anzunehmen, daß eine formale Spezifikationstechnik mit hohem mathematischem Gehalt sich in der Praxis durchsetzen wird, wenn damit nur Dokumente für das Bücherregal produziert werden und eine direkte Überleitung in die Programmierphase nicht erfolgen kann. Das funktioniert nicht einmal in universitärem Kontext [15]. Eine Spezifikationstechnik muß deshalb verschiedene Abstraktionsstufen anbieten, wovon sich die niederen automatisch in Code übersetzen lassen.

10. Konstruktive algebraische Spezifikationen sind im wesentlichen dasselbe wie Programme einer funktionalen Programmiersprache.

11. Deshalb sind effiziente Verfahren zur Implementierung funktionaler Programmiersprachen [5, 11, 17] von besonderer Bedeutung (s.a. These 17).

12. Funktionale Programmiersprachen sind wegen ihrer freundlichen Eigenschaften in Bezug auf Verifikation und Optimierung von besonderer, vielfach unterschätzter Bedeutung. Eine Ausbildung in den Grundlagen der Programmierung braucht deshalb eine Darstellung der Grundlagen der funktionalen Programmierung [18].

13. Trotzdem gibt es Bereiche, die sich leichter imperativ programmieren lassen oder die etwa nach objektorientierten oder Logik-Programmiersprachen verlangen. Statt eine Medizin gegen alle Krankheiten zu propagieren, ist daher eher eine *multiparadigmatische Programmierung* [14] erfolgversprechend, bei der man für jedes Modul die Methodik und Sprache auswählen kann, die hierfür am besten paßt. N.B.: Sog. multiparadigmatische Programmiersprachen sind selten in sich konsistent und konzeptuell sauber. Beliebiges Vermischen von Paradigmen führt vielmehr dazu, daß diese völlig unscharf werden.

14. Beweisbar korrekte Programme verlangen eine Programmiersprache, deren Semantik ebenfalls präzise festgelegt ist. Das Modula-2-Normungsdokument [1] der ISO-Arbeitsgruppe WG13, in der ich selbst mitarbeite, ist historisch der erste Versuch, eine formale Semantik (hier in VDM-SL formuliert) zum Bestandteil der internationalen Norm einer Programmiersprache zu machen. Die formale Semantik hat zwei Stoßrichtungen: sie hilft dem Programmierer, unklare semantische Fragen auf andere Weise als durch Ausprobieren mit einem bestimmten Compiler zu beantworten, und sie ist die Grundlage für die Entwicklung korrekter Compiler. (Der Compiler aus [4] wurde allerdings entwickelt, bevor die ISO-Norm entstand.)

15. Wenn die Korrektheit eines Programms bewiesen ist, müssen wir uns als nächstes über die Effizienz Gedanken machen. Dazu muß man notwendigerweise wissen, wie das Programm auf der Maschinenebene aussehen wird, welche Optimierungen ein Compiler vornehmen kann und wird und was man vom Compiler nicht erwarten kann. Grundkenntnisse in Compilerbau [13] gehören daher zur Ausbildung jedes Informatikers. Programmierwerkzeuge (s.a. These 16) weisen auf die Stellen hin, wo Effizienzbetrachtungen besonders gefragt sind.

16. Programmieren ist nicht nur eine Ingenieurskunst, sondern hat auch Aspekte eines Handwerks. Ein Handwerker braucht gute Werkzeuge; Programmierer sind deshalb im Gebrauch von Software-Werkzeugen zu schulen und müssen ausgebildet werden, selbst Werkzeuge zu schreiben (z.B. [7, 12]).

17. Auch zur Spezifikation gehören Werkzeuge: z.B. kann die Konsistenzprüfung einer konstruktiven algebraischen Spezifikation mit einer Gleichungsspezifikation durch ein Werkzeug unterstützt werden [16]; andere Werkzeuge [6, 9] erlauben das Experimentieren („Frage-und Antwort-Spiel") mit einer Spezifikation. Vermutlich das wichtigste Werkzeug in diesem Zusammenhang setzt konstruktive algebraische Spezifikationen um in effizienten iterativen Maschinencode [5, 8, 14].

18. Aus all dem ergibt sich das folgende Vorlesungsprogramm in Programmierung:

    Informatik I („Vom Problem zum Programm") Spezifikation, Algorithmus, Verifikation, Rechenaufwand, Abstrakte Maschinen, Pascal, Abstrakte Datentypen

    Informatik II („Vom Algorithmus zur Maschine") Rechnerstruktur, Prozessor-Architektur, Binärarithmetik, Laufzeitorganisation, Assemblerprogrammierung, Binder, Lader, Betriebssystem

    Informatik III („Einführung in die Theoretische Informatik") Formale Sprachen, Automatenmodelle, Berechenbarkeit, Komplexitätstheorie

    Algorithmen Suchen, Sortieren, Graphenalgorithmen, konkrete Komplexität, Effizienzsteigerung, Divide-et-impera, dynamische Programmierung, heuristische Ansätze zu NP-vollständigen Problemen

    Softwaretechnik Software-Lebenslauf, Methoden zur Anforderungsanalyse und -definition, Systementwurf, Modulkonzept, Datenkapselung und Objektorientierung, Software-Werkzeuge, Testen und Messen von Software

    Semantik von Programmiersprachen Mathematisch-logische Grundlagen, operationelle und axiomatische Semantik, denotationelle Semantik, VDM-SL

    Grundlagen der funktionalen Programmierung Polymorphie, Typinferenz, Funktionen höheren Typs, Auswertungsstrategien, Laufzeitorganisation

Compilerbau Syntaxanalyse, Laufzeitsysteme, Codeerzeugung, Optimierung, „Bootstrapping“ von Compilern

Konzepte von Programmiersprachen Programmierparadigmata (imperativ, funktional, logisch, objektorientiert), Interpretersprachen, Compilersprachen, gemeinsame und individuelle Konzepte

Algebraische Software-Spezifikation Universelle Algebra, Spezifikationen und Modelle, Gleichungskalkül, initiale und operationelle Semantik, parametrisierte Spezifikationen, konstruktive algebraische Spezifikationen

19. In der Forschung ist die Thematik wie folgt vorgezeichnet:

Implementierung funktionaler Programmiersprachen Weitere Verbesserung der Übersetzungsschemata, Ausweitung des Einsatzbereichs, Integration in reale Compiler

Software-Entwicklungsumgebung Integration der Werkzeuge zur Behandlung und Transformation von Spezifikationen, konsistente Benutzerschnittstelle

Programmvisualisierung und Visuelle Programmierung Werkzeuge zur Ableitung graphischer Darstellungen aus Modultexten und zur graphischen Manipulation davon, Integration in die Software-Entwicklungsumgebung

Programmierumgebung Perfektionierung und Vereinheitlichung der vorhandenen Programmier-Werkzeuge, Entwicklung neuer Werkzeuge

Literatur

[1] ISO SC22/WG13: *Modula-2—Draft International Standard: DIS 10154*. British Standards Institution, June 1994.

[2] KLAEREN, HERBERT: *Algebraische Spezifikation — Eine Einführung*. Springer Verlag, Berlin-Heidelberg-New York, 1983.

[3] KLAEREN, HERBERT: *A Constructive Method for Abstract Algebraic Software Specification*. Theoretical Computer Science, 30:139–204, 1984.

[4] KLAEREN, HERBERT: *Celerity Modula-2 Compiler Reference Manual*. Technischer Bericht Universität Tübingen, 1988.

[5] KLAEREN, HERBERT: *Ein algebraischer Ansatz zur Rekursionselimination*. Habilitationsschrift, RWTH Aachen, 1988.

[6] KLAEREN, HERBERT: *ModAs/86 User Manual*. Technischer Bericht Universität Tübingen, 1988.

[7] KLAEREN, HERBERT: *Celerity Modula-2 Utilities Reference Manual*. Technischer Bericht Universität Tübingen, 1989.

[8] KLAEREN, HERBERT: *Embedding Functionally Described Abstract Data Types into Modula-2 Programs*. In: *First International Modula-2 Conference*, Seiten 115–119, Bled, 1989.

[9] KLAEREN, HERBERT: *ModAs/86 — Eine funktionale Programmierumgebung*. In: *Workshop Programmierumgebungen für funktionale und logische Sprachen*, Seiten 147–151, Bad Honnef, 1990.

[10] KLAEREN, HERBERT: *Vom Problem zum Programm — Eine Einführung in die Informatik*. Teubner Verlag, 1990.

[11] KLAEREN, HERBERT: *Efficient Implementation of a Functional Programming Language*. Arbeitspapier, Universität Tübingen, Feb. 1991.

[12] KLAEREN, HERBERT: *Some Elements of a Modula-2 Development Environment*. In: *Second International Modula-2 Conference*, Seiten 225–233, 1991.

[13] KLAEREN, HERBERT: *Übersetzung einer einfachen Programmiersprache*. Arbeitspapier, Universität Tübingen, Sep. 1992.

[14] KLAEREN, HERBERT und PETER THIEMANN: *A Clean Modula-2 Interface to Abstract Data Types*. Structured Programming, 11:69–77, 1990.

[15] LUDWIG, ANDREA: *Bericht über das Praktikum Software-Engineering WS 1982/83, SS 1983*. Berichte des Lehrstuhl für Informatik II 9, RWTH Aachen, 1983.

[16] PETZSCH, HEIKO: *Implementierung algebraischer Spezifikationen in Prolog zur Unterstützung interaktiver Induktionsbeweise*. Schriften zur Informatik und angewandten Mathematik 125, RWTH Aachen, 1987.

[17] THIEMANN, PETER: *Konzepte zur effizienten Implementierung strukturell-rekursiver Programme*. Doktorarbeit, Universitat Tubingen, 1991.

[18] THIEMANN, PETER: *Grundlagen der funktionalen Programmierung*. Teubner, 1994.

[19] WIRTH, NIKLAUS: *Gedanken zur Software-Explosion*. Informatik-Spektrum, 17(1):5–10, 1994.