Eingebettete Software – a „terrifying growth of complexity“?

Dr. Axel Sodtalbers

www.softwarehaftung.de

Bochum, 14. Juli 2010

RUHR-UNIVERSITÄT BOCHUM

FAKULTÄT FÜR ELEKTROTECHNIK UND INFORMATIONSTECHNIK

ETI

RUB

Übersicht

Niklaus Wirth

⋆ 1934 in Winterthur, Schweiz

ab 1968 Professor an der ETH Zürich

1976/77 Xerox Palo Alto Research Center (Xerox PARC)

1984 ACM, A.M. Turing Award

1999 emeritiert.

Niklaus Wirth als Entwickler von Programmiersprachen

Pascal
- Strukturierte Programmierung

Modula-2
- Modulare Programmierung
- Lilith Workstation (1980); Hardware, Betriebssystem, Compiler, grundlegende Applikationen

Oberon
- Objekt-orientierte Programmierung
- Oberon-System (Betriebssystem); Ceres Workstation
- Leitmotiv:

  „Make it as simple as possible, but not simpler. (A. Einstein)“ [Oberon08]

Niklaus Wirth

Zitat [Skulski2004] über Niklaus Wirth:

Component Pascal: historical perspective

- How an electrical engineer changed the computer science.
  - Electrical engineers know that circuits should not burn or explode.
  - Computer scientists assumed that if SW explodes, they can reboot.
  - An electrical engineer said “stop the madness”. His name: Prof. Niklaus Wirth.

N. Wirth 1995 über Softwarekomplexität

Aufsatz „A Plea for Lean Software“, 1995 [Wirth95]

Gründe für „Fat Software“ (1):

- „Featuritis“
  - Neues Release muss neue Features haben.
  - Was der Kunde wünscht, wird implementiert – ohne „Rücksicht auf Verluste“.
- keine inhärente, sondern selbst auferlegte Komplexität
- Komplexität hilft beim Verkauf:
  - Abnehmer denkt: was komplex ist, muss mächtig sein.
  - Verkäufer: Kundenbindung; Zusatzgeschäft (Serviceverträge, Beratung, Schulung, . . .)

> *„Customer dependence is more profitable than customer education.“*

N. Wirth 1995 über Softwarekomplexität

Gründe für „Fat Software“ (2):

- Es fehlen klare Konzepte.
  - Anzeichen: Umfang der Handbücher / Hilfen.
  - Auch bei Programmiersprachen.
- Gute Konzepte, gutes Design ist schwierig, zeitraubend und teuer.
  - Hardware: Fortschritt beruht auf verbesserte Fertigung (Fabrikation), nicht auf Design.
  - Software: Fortschritt kann nur auf Design beruhen (Fabrikation bedeutungslos).

N. Wirth 1995 über Softwarekomplexität

Gründe für „Fat Software“ (3):

- Zeitdruck.
- Methoden aus der Wissenschaft werden von der Industrie ignoriert.

> *„To reduce software complexity by concentrating only on the essential is a proposal swiftly dismissed as ridiculous in view of customers's love for bells and whistles. When ‚everything goes‘ is the modus operandi, methodologies and disciplines are the first casualities.“*

Kritik 2008 im Grundsätzlichen bestätigt

In [Wirth08, S. 7 f.] führt N. Wirth u.a. aus:

- Softwareengineering habe zwar von neuen technisch ausgefeilten („sophisticated“) Entwicklungswerkzeugen profitiert, . . .
- . . . aber bei der Qualität der Produkte spiegele sich das kaum wider.

> „*After all, the increase of power was itself the reason for the terrifying growth growth of complexity.*“

Und wie ist es bei eingebetteter Software?

These:

Die von Wirth beschriebene Situation
wird auch bei eingebetteter Software eintreten.

Anzeichen:

1. Umfang eingebetteter Software wächst bereits jetzt stark.
2. „Featuritis“ bereits erkennbar.
3. Programmiersprache C immer noch im Einsatz.
4. Fehlerhafte Software auf rund 30.000.000 EC-Karten.

Another terrifiying growth of complexity?

Anzeichen: Umfang eingebetteter Software wächst stark.

Beispiel: Automobilelektronik [Bitkom2010]:

> *„Autos haben heute 100 MByte Software in ihren Rechnern laufen, mit einer Komplexität, die schneller wächst als jene von IT-Systemen von SAP, Oracle und Microsoft zusammen. [. . .]*
>
> *Obwohl diese Zahlen vergleichbar sind mit jenen der weltweit größten Software-Pakete, wie z. B. Microsoft Windows, ist die Komplexität eingebetteter Systeme bei weitem größer. [. . .]“*

Grund: zusätzliche Anforderungen hinsichtlich

- Verfügbarkeit
- Robustheit und
- Echtzeitfähigkeit.

Another errifiying growth of complexity?

Anzeichen: „Featuritis“ bereits erkennbar.

- Immer neue Funktionen werden implementiert
- Paradebeispiel: Kraftfahrzeug

  Beispiel: das „vernetzte Breitband-Auto“ (Heise-Online 3/2010); eCall (EU); Auto mit Internetzugang und/oder SIM-Karte

Another terrifiying growth of complexity?

Anzeichen: Programmiersprache C immer noch im Einsatz.

[Bautsch07]:

> *„Outdated programming languages are still in use today and a pithy comment made by the famous Swiss computer scientist, Niklaus Wirth, is appropriate in this context:* ‚It is indeed absolutely surprising with which equanimity the notational monster C was accepted by the world-wide programmer's community.‘ *There can really be no debate about this because, for example, there are still job adverts asking for programming skills in C for automotive security systems today. Who really wants to drive a car with such an outdated security system?“*

Another terrifiying growth of complexity?

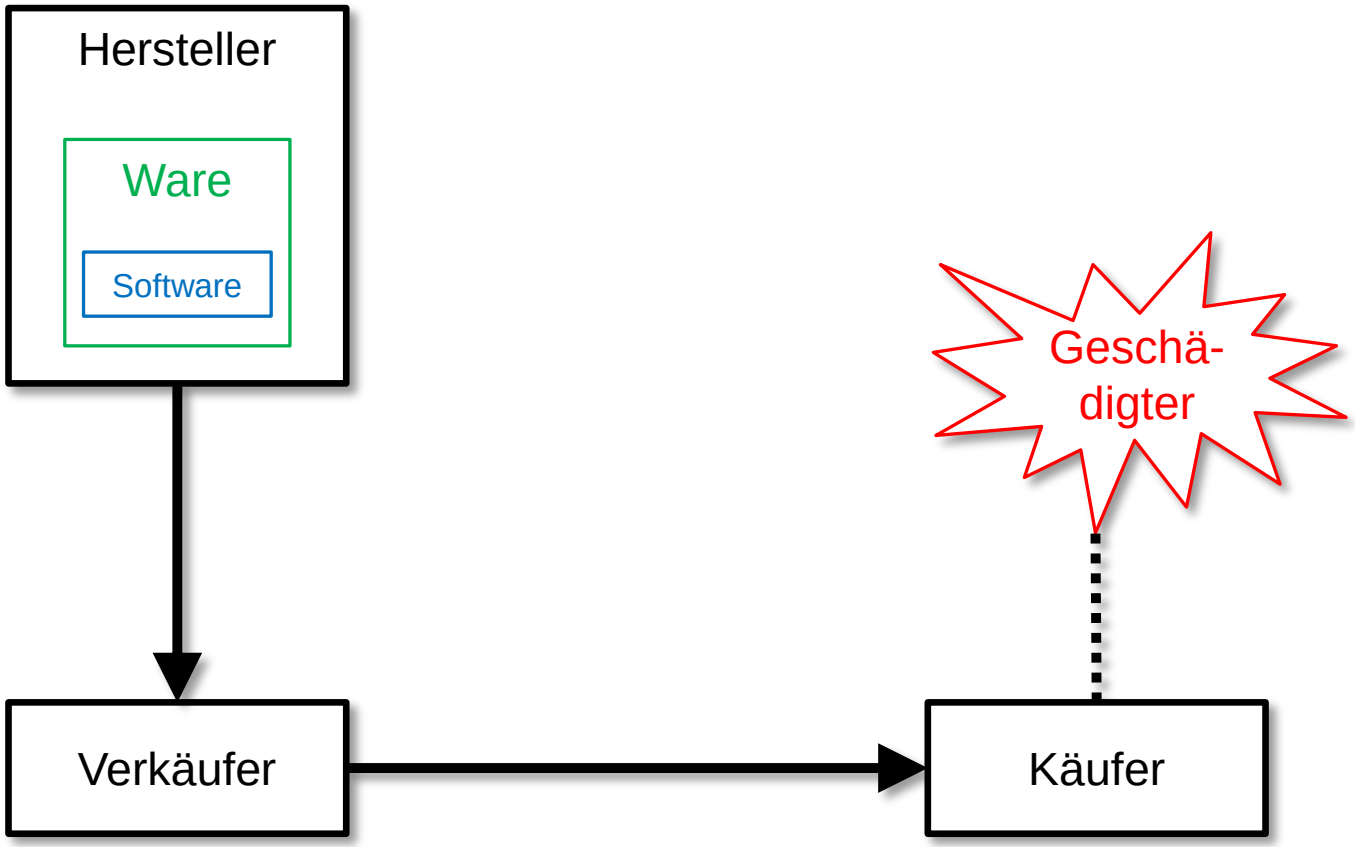
Anzeichen: Fehlerhafte Software auf rund 30.000.000 EC-Karten.

- „Jahr-2010-Fehler“
- Enormer finanzieller Schaden („. . . bis zu 300 Millionen Euro . . . “)
- Grund für Fehler wohl auch: komplexer Implementierungsstandard.

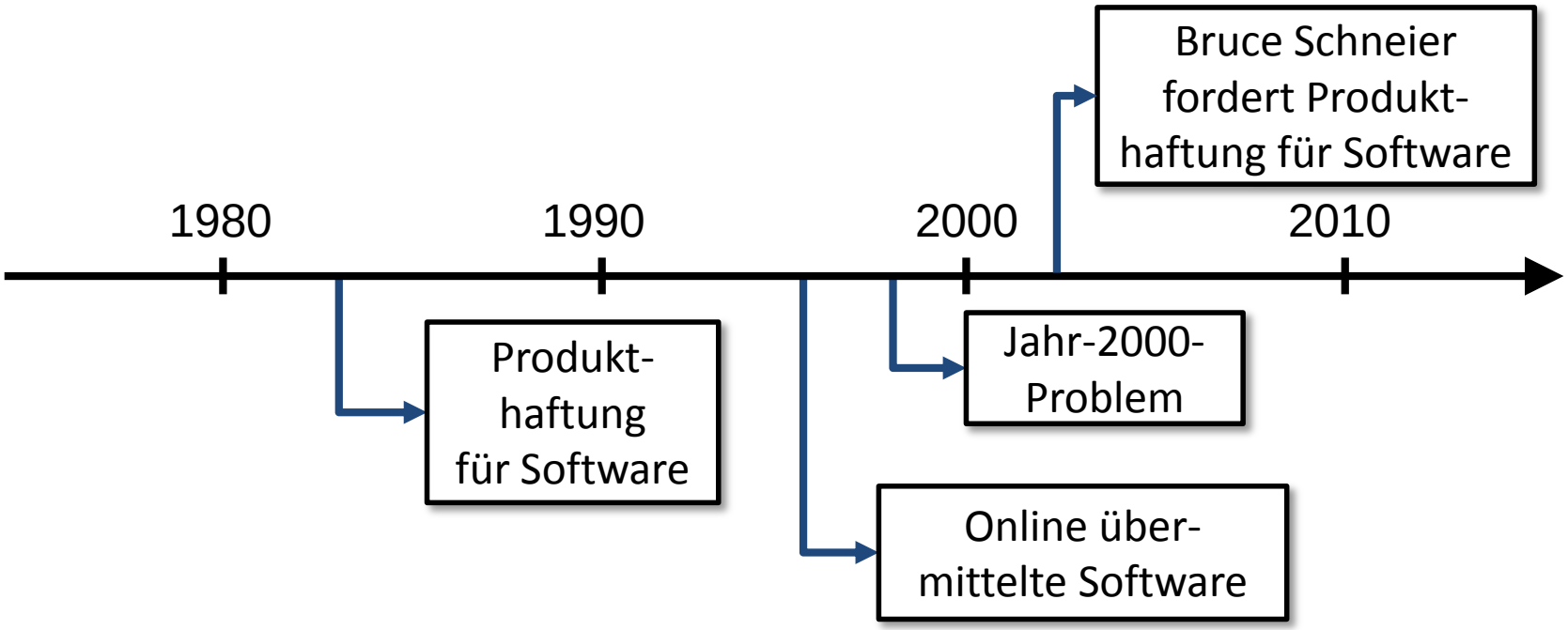
Bewertung aus der juristischen Perspektive

Wie ist das aus der juristischen Perspektive zu bewerten?

Lieferbeziehungen – Vertrag oder kein Vertrag?



Juristische Diskussion um die außervertragliche Haftung



Heute:

- „Software überall“.
- Stückzahlen, die früher schlicht unvorstellbar waren.

Anspruchsgrundlagen

- § 823 Absatz 1 BGB
- § 1 Produkthaftungsgesetz (ProdHaftG)
- § 823 Absatz 2 BGB i.V.m. Schutzgesetz
  - Geräte- und Produktsicherheitsgesetz (GPSG)
  - Strafgesetzbuch (StGB)
  - Medizinproduktegesetz (MPG)
  - ...

§ 823 Absatz 1 BGB

- Verkehrspflichten
  - Konstruktionspflichten
  - Fabrikationspflichten
  - Instruktionspflichten
  - Produktbeobachtungspflichten (aktiv und passiv)
- Je höher der Rang des bedrohten Rechtsguts, desto größere Anstrengungen für Fehlerfreiheit.
- Standards- und Normen sind (im Außenverhältnis) in aller Regel irrelevant.
- Abhängig vom Verschulden: außer Acht lassen der im Verkehr erforderlichen Sorgfalt (§ 276 BGB).
  Aber: → Beweislastumkehr!

BUNDESGERICHTSHOF

IM NAMEN DES VOLKES

Produkthaftungsgesetz (ProdHaftG)

- Unabhängig vom Verschulden
- Anwendung auf eingebettete Software umstritten, aber nach überwiegender Ansicht zu bejahen.
- Keine Haftung, wenn Fehler nach Stand von Wissenschaft und Technik nicht erkannt werden konnte. (Beweislast: Hersteller!)
- Keine Produktbeobachtung; kein Rückruf.
- Auch Quasi-Hersteller; Importeure; u. U. Händler.
- Teilehersteller – eingebetteter Software – kann wie Endhersteller haften!

Haftungsausschluss, weil Softwarefehler unvermeidlich sind?

? Fehler sind unvermeidbar, weil Programme zu komplex sind.
Also ist eine Haftung für solche Fehler nicht angemessen.

Kein Haftungsausschluss

Diese Argumentation trifft nicht zu:

- Gleicher Befund gilt auch für andere Produkte (Bau; Auto).
- Dogmatisch nicht haltbar:
  - ProdHaftG: gerade vom Verschulden abhängig
  - ProdHaftG: Sicherheitserwartungen wirklich dahin gehend?
  - BGB: Beweis des fehlenden Verschuldens wird nicht gelingen.

Wenn Fehler wirklich unvermeidbar sind, das ist das doch erst recht ein Grund, den Hersteller haften zu lassen!

Kein Haftungsausschluss

Sind denn Fehler wirklich unvermeidbar?
N. Wirth nennt u.a. folgende Gründe für die Komplexität:

- „Featuritis“.
- Komplexität als „Verkaufshilfe“ (Kundenbindung).
- Klare Konzepte fehlen.
- Gutes Design ist schwierig und teuer.
- Zeitdruck.
- Ignoranz gegenüber Methoden aus der Wissenschaft.

Das sind gerade keine Gründe, die rechtlich für einen Haftungsausschluss sprechen – im Gegenteil.

Kein Haftungsausschluss

Botschaft von N. Wirth: Komplexität ist vermeidbar

Problem ist nicht die inhärente, sondern die selbst auferlegte Komplexität.

“But it is not the inherent *complexity that should concern us; it is the* self-inflicted *complexity.“*

[Wirth95, S. 65]

Herzlichen Dank für Ihre Aufmerksamkeit!

Download
der aktuellen Präsentationsunterlagen
unter

www.softwarehaftung.de

Kontakt

Dr. Axel Sodtalbers
Rechtsanwalt

Brookweg 120b
26127 Oldenburg

info@softwarehaftung.de

www.softwarehaftung.de

Quellen I

Wirth, Niklaus,
A Plea for Lean Software, IEEE Computer, 28, 2, (Feb. 1995),
S. 64 ff.,
zit. als [Wirth95].

Wirth, Niklaus,
A Brief History of Software Engineering, 25.2.2008,
zit. als [Wirth08].

Wirth, Niklaus,
The Programming Language Oberon, Revision 1.10.90,
zit. als [Oberon08].

Quellen II

Skulski, Wojtek,
BlackBox Component Builder For Scientists and Engineers,,
to be presented at LLE in February 2004,
Laboratory for Laser Energetics, University of Rochester, New York,
zit. als [Skulski2004].

Bitkom (Hrsg.),
Eingebettete Systeme – Ein strategisches Wachstumsfeld für Deutschland, 2010,
zit. als [Bitkom2010].

Quellen III

Markus Bautsch,
Cycles of Software Crises, How to avoid insecure and uneconomic software, in: ENISA Quarterly Vol. 3, No. 4, Oct-Dec 2007, Seite 3 ff., zit. als [Bautsch07].

Bruce Schneier,
Liability and Security, in: Crypto-Gram Newsletter 15. April, 2002