Softwarequalität – Geschichte und Trends

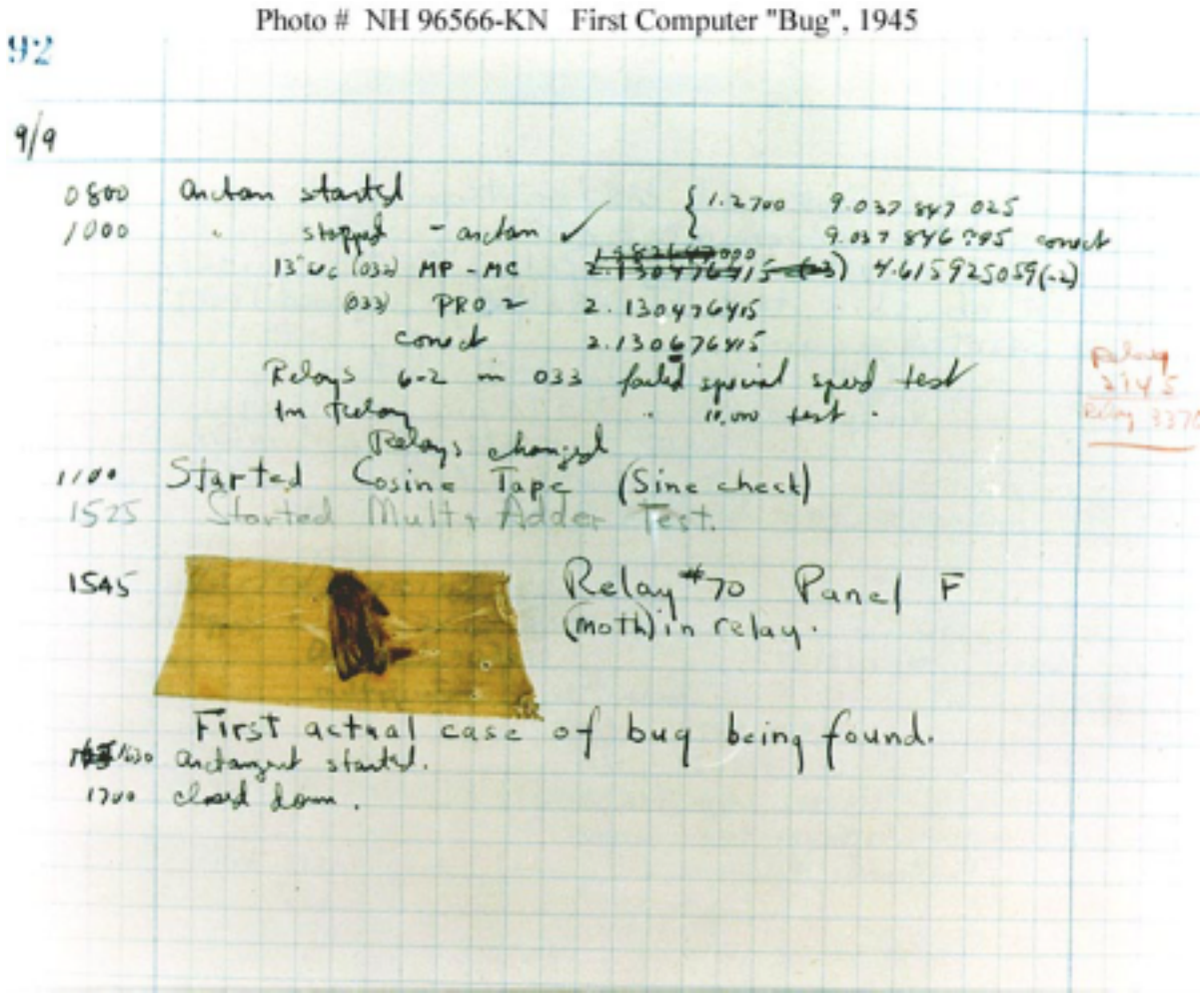
Prof. Dr. Holger Schlingloff

Humboldt-Universität zu Berlin
und
Fraunhofer FIRST

Überblick

- **Verifikation**
  - historische Softwarefehler, Beweiskalküle, Programmbeweise
- **Spezifikation**
  - Temporallogik, Modellprüfung
- **Testtheorie**
  - Testverfahren, Testgenerierung

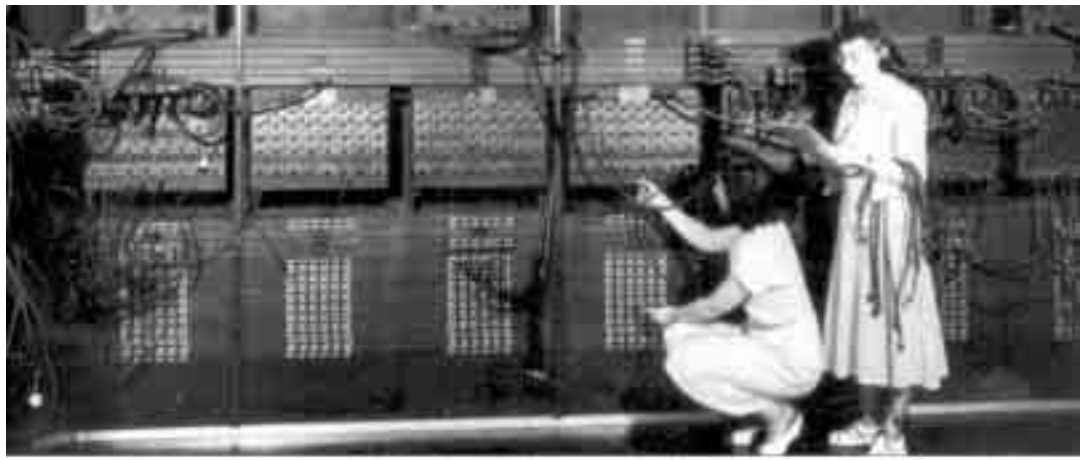
der „erste Bug“



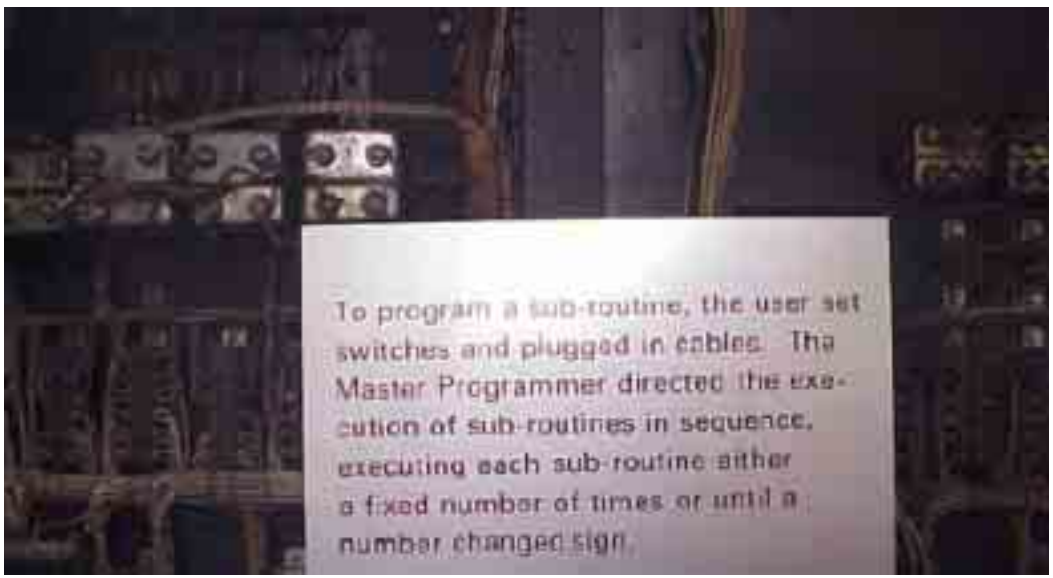
www.history.navy.mil/photos/images/h96000/h96566k.jpg
Information Page: www.history.navy.mil/photos/pers-us/uspers-h/g-hoppr.htm

- Harvard University Mark II Aiken Relay Calculator
- Wort bereits vorher im Gebrauch! (Telegraphie)
- Smithsonian Museum

Programmierfehler?



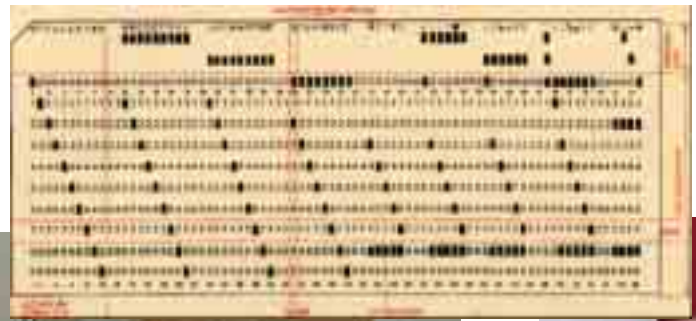
http://twiki.im.ufba.br/pub/MAT052/WebHome/prog-lang-hist.pdf



http://www.columbia.edu/acis/history/eniac.html

2. Set Accumulator 8 to clear by removing all cables from it.
3. Set Accumulator 7 to take care of the dummy program.
(a) Attach Program Line 1-3 to program input terminal 5i.
(d) Set the Repeat Switch for Program Control 5 to 1.
...
7. **Clear the Eniac.**
8. **Start the Eniac.**

Strich- und Lochkarten



- Strichkarten: Fehler (?) durch Graphitstaub
- Lochkarten: Stapelbetrieb zwingt zu sorgfältiger Arbeit

Intelligente Fehler?

- Alan Turing, „Computing Machinery and Intelligence“ (1950):
  - Q: Please write me a sonnet on the subject of the Forth Bridge.
  - A : Count me out on this one. I never could write poetry.
  - Q: Add 34957 to 70764.
  - A: (Pause about 30 seconds and then give as answer) 105621.
  - Q: Do you play chess?
  - A: Yes.
  - Q: I have K at my K1, and no other pieces. You have only K at K6 and R at R1. It is your move. What do you play?
  - A: (After a pause of 15 seconds) R-R8 mate.
- „I believe that in about fifty years‘ time it will be possible to program computers, with a storage capacity of about 10^9, to make them play the imitation game so well that an average interrogator will not have more than 70 percent chance making the right identification after five minutes of questioning.“

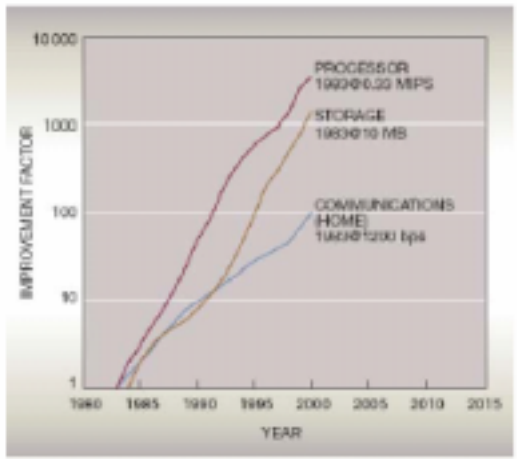
Die Softwarekrise

- Software wird erst ab 1968 ernst genommen
- Edsger W. Dijkstra (1972): „The major cause [of the software crisis] is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem. In this sense the electronic industry has not solved a single problem, it has only created them...“

Warum ist SW-Qualität wichtig?

- Moore‘s Law: Alle 18 Monate Verdoppelung der Möglichkeiten (Raum / Zeit)
- Programmierer verhalten sich wie ein ideales Gas (füllen jeden Raum bis an die Grenzen)

Verbesserung in den letzten 25-30 Jahren



→ C. Freytag

- Modularisierung und Wiederverwendung
- Computer ersetzen hergebrachte Technologien
- Computer steuern mehr und mehr alltägliche Dinge
- Computer eröffnen neue Märkte (e-irgendwas)

Berüchtigte Softwarefehler

- Therac-25 Bestrahlungsgerät – 1985/86
- Ariane 5 Jungfernflug – 24.12.1979
- AT&T-Ferngesprächsystem – 15.1.1990
- Patriot-Luftabwehrrakete – 25.2.1991
- Ölbohrplattform Sleipner – 23.8.1991
- Pentium-Bug – 19.10.1994
- Stellwerk Hamburg Altona – 24.3.1995
- Mars Polar Lander – 28.3.2000
- Parteitag der Grünen – 24.2.2002
- ...

Dijkstras Antwort

- "Program testing can be used to show the presence of bugs, but never to show their absence!" (Notes On Structured Programming, 1969)
- „we ... take the position that it is not only the programmer‘s task to produce a correct program but also to demonstrate ist correctness in a convincing manner“

Formale Verifikation

- Temporallogik
  - A. Pnueli (1977): The temporal logic of programs
- Hoare-Logik
  - C.A.R. Hoare (1969): An axiomatic basis for computer programming
- Partielle Korrektheit
  - R. W. Floyd (1967): Assigning meanings to programs
- Invarianten
  - P. Naur (1966): Proof of algorithms by general snapshots
- Beweise
  - J. von Neumann (1947/51): Planning and coding of problems for an electronic computing instrument
- Assertions
  - A. Turing (1949): Checking a large routine






Turings Ansatz

3-Seiten-Papier

How can one check a routine in the sense of making sure that it is right? In order that the man who checks may not have too diffcult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

…

- a dashed letter indicates the value at the end of the process represented by the box.
- an undashed letter represents the initial value of a quantity.

…

In this case the claim is that if we start with control in condition *A* and with *n* in line 29 we shall find a quantity in line 31 when the machine stops which is *n*!

…

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

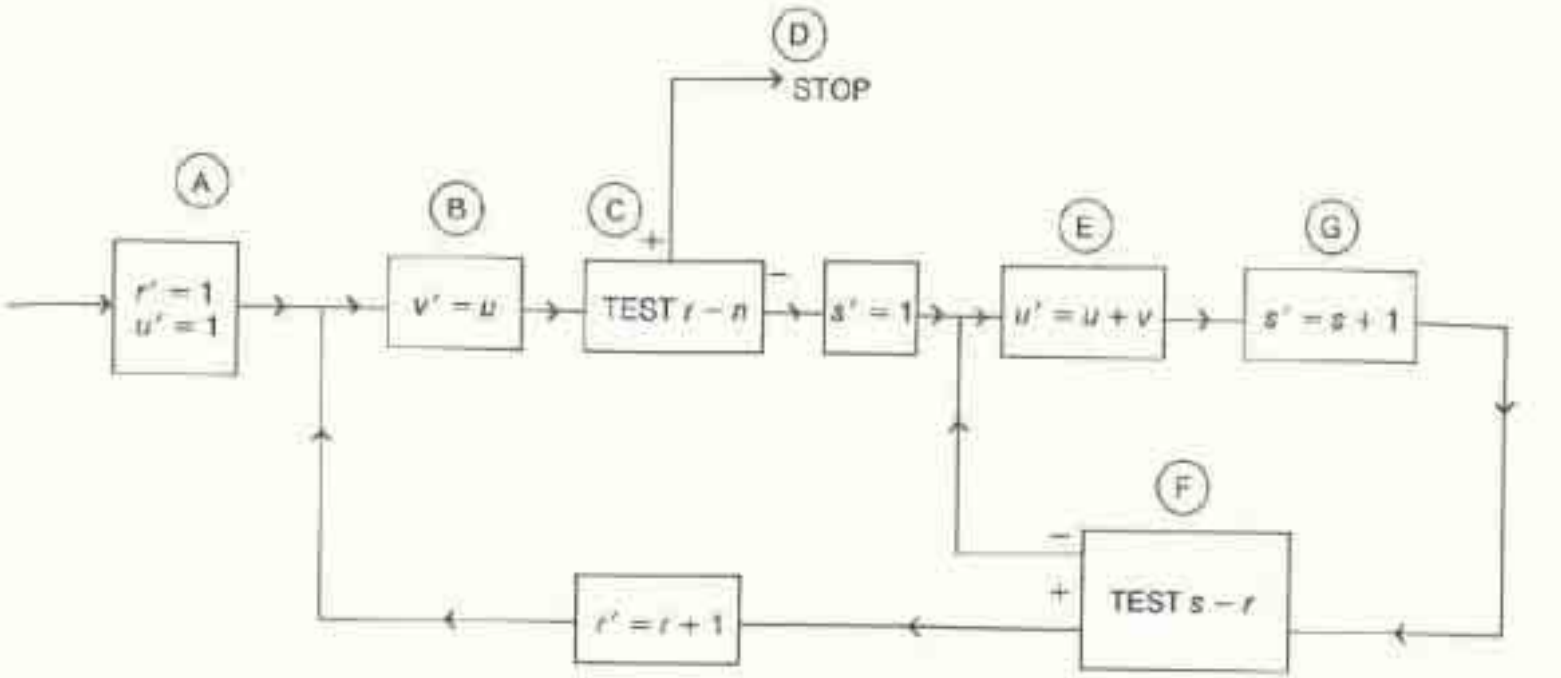


Figure 1 (Redrawn from Turing's original)

STORAGE LOCATION	(INITIAL) (A) $k=6$	(B) $k=5$	(C) $k=4$	(STOP) (D) $k=0$	(E) $k=3$	(F) $k=1$	(G) $k=2$
27					s	$s+1$	s
28		r	r		r	r	r
29	n	n	n	n	n	n	n
30		$\underline{r}$	$\underline{r}$		$s\|\underline{r}$	$(s+1)\|\underline{r}$	$(s+1)\|\underline{r}$
31			$\underline{r}$	$\|n$	$\underline{r}$	$\underline{r}$	$\underline{r}$
	TO (B) WITH $r'=1$ $u'=1$	TO (C)	TO (D) IF $r=n$ TO (E) IF $r<n$		TO (G)	TO (B) WITH $r'=r+1$ IF $s \geq r$ TO (E) WITH $s'=s+1$ IF $s<r$	TO (F)

Figure 2 (Redrawn from Turing's original)

Keine Silberkugel?

- Viele konkurrierende formale Methoden
  - EN50128: “Several examples of formal methods are described in the following subsections. The formal methods described are CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM and Z.”
  - Keine Methode für alle Anwendungen gleichermaßen geeignet
- Schwierigkeitsgrad eines Beweises ist mindestens 3-5 mal so hoch wie der des entsprechenden Programms
  - Produktivitätsmaße z.B. 20 Zeilen/Personentag
  - Aufwand wird oftmals gescheut bzw. rechnet sich nicht
- Formale Verifikation wird nur in sicherheitskritischen, „überschaubaren“ Projekten eingesetzt
  - Cenelec SIL4 Systeme, z.B. Bahncomputer-Voter
  - Bird Satelliten-Betriebssystem
  - Keno Lotto-Ziehungsgerät

II. Spezifikation

- Philosophische Frage: „Was ist ein Fehler?“
  - → nicht erwartungskonformes Verhalten oder Beschaffenheit eines Artefaktes
  - wer hat die Erwartungen? („bug or feature?“)
  - nur bezüglich einer Spezifikation (explizit oder implizit) definierbar
- Praktische Frage: „Wie spezifiziert man SW?“
  - „Spezifikation“ aus l. *species* "Art, Gestalt" und l. *facere* "machen"
  - Wikipedia: Text, der die Syntax und Semantik (zum Beispiel einer Software in der Informatik) oder die Implementierung eines bestimmten Bestandteiles oder Produktes beschreibt, dass heißt eine deklarative Beschreibung, was etwas ist oder bewerkstelligt. Anhand einer guten Spezifikation lässt sich demnach ein detaillierter Vergleich zwischen Soll- und Ist-Zustand vornehmen.
  - Beschreibung in natürlicher Sprache?

Notwendigkeit der Formalisierung

- Natürliche Sprache mehrdeutig!
  - Das Mädchen lobt die Mutter
  - Das sagten die Kinder der Tante
  - Der Junge beobachtet den Mann mit dem Fernglas
  - Ich denke über die Aufgabe meines Jobs nach
  - Der Ausbau der PC-Zusatzkarte ist erforderlich
  - Die SPD geht mit der Zeit.
- SW-Beispiel:

„alle 30 Sekunden sollen die Werte der Sensoren abgelesen werden; wenn die Standardabweichung 0,25 überschreitet, soll die Normalisierungsprozedur ausgeführt werden, anschließend sollen die Werte an das Analysepaket weitergegeben werden.“

Akzeptanztest ergibt falsche Analysewerte.

Problem: Komma!

☺ Beispiele für Zeichensetzungs-Probleme

- Komma zuviel?
  - Der brave Mann denkt an sich selbst zuletzt.
  - Der brave Mann denkt an sich, selbst zuletzt.
- Kommaverschiebung?
  - Treu war sie, nicht ohne Tränen ließ ich sie gehen.
  - Treu war sie nicht, ohne Tränen ließ ich sie gehen.
- PL/1:
  - DO k = 1 TO 100 WHILE(a = 3); ... ; END;
  - DO k = 1 TO 100, WHILE(a = 3); ... ; END;
- Fortran
  - DO 5 K = 1. 3 ... 5 CONTINUE wird interpretiert als Zuweisung
- C ist nicht besser:
  - while(x > 0,1) { ... }

http://www-aix.gsi.de/~giese/swr/sprache4.html

Logik als Spezifikationssprache

- Aristoteles: *Formalisierung menschlichen Denkens*
  - „Alle Menschen sind sterblich, Sokrates ist ein Mensch, also ist Sokrates sterblich.“
  - Beschreibung von Sachverhalten und Schlussweisen
- Mathematische Logik
  - Hilbert, Russell, Frege, Gödel, ...
  - Beschreibung mathematischer Sachverhalte
  - Adäquat so lange SW (nur) mathematische Funktionen berechnete

Temporale Logik

- Formalisierung von *Vorstellungen über die Zeit*
  - „Auf jeden Regen folgt auch wieder Sonnenschein.
    In der Hölle scheint nie die Sonne.
    Also regnet es in der Hölle nicht!“

➔ Wurzeln der temporalen Logik bis Aristoteles rückverfolgbar

➔ moderne Temporallogik beginnt mit A. Prior: „Past, Present and Future“ 1967

- Formalisierung von Sätzen wie „Auf jeden Regen folgt auch wieder Sonnenschein“ mit so genannten *Modaloperatoren*
  - □ „always“
  - ◇ „sometime“

$$\Box(\text{rain} \rightarrow \Diamond \text{sunshine})$$

- Deduktionssysteme
  - $\Box(\text{rain} \rightarrow \Diamond \text{sunshine}), \text{hell} \rightarrow \Box\neg\text{sunshine} \vdash \text{hell} \rightarrow \Box\neg\text{rain}$

- Die späten 1970-er: Zusammenhang zur Informatik
  - V. Pratt, „dynamic logic“ (1976)
  - F. Kröger, „logic of algorithmic reasoning“ (1976)
  - A. Pnueli, „The temporal logic of programs“ (1977)

- Beschreibung der Semantik von Programmen
  - Beispiel: m_0: while x<5 do x:=x+1 od
    wird zu $\Box$ (**at** $m_0 \wedge x<5 \rightarrow x' = x+1 \wedge \bigcirc$ **at** m_0)
- Formalisierung von Korrektheitsaussagen über Programme
  - Beispiel: $\Box \neg$(**at** $m_0 \wedge$ **at** m_1)
  - Theorie von Sicherheits- und Lebendigkeitsaussagen
- Interaktive Programmverifikation
  - Erweiterung der Hoare-Logik für parallele Programmierung
  - Unterstützende Beweissysteme
  - wenig praktische Relevanz

- Ab Mitte der 1980-er: vollautomatische Verifikation, Modellprüfung
  - Semantische Relation ||- definiert, ob eine Formel in einem Modell gilt
  - E.M. Clarke, Synthesis of synchronization sceletons...
  - Modell = Programm (-ausführung), Formel = Programmeigenschaft

- Algorithmische Theorie des Model Checking
  - Komplexität für verschiedene Varianten
  - Zusammenhang zur Automatentheorie
  - K. McMillan: BDDs als effiziente Datenstrukturen (1989)

- Bislang erfolgreichster Zweig der Temporallogik
  - in der Hardware-Verifikation inzwischen obligatorischer Standard
  - auch für Software eingesetzt

- Die 1990er: Verfeinerungen und Abstraktionen
  - TLA (Lamport): Trennung zwischen operationalem und nichtoperationalem Anteil; Spezifikation „wirklicher Systeme“
  - Implementierungsrelationen zwischen Spezifikationen
  - Realzeiterweiterungen, Duration Kalkül
  - Raum-Zeit-Logiken
  - hybride Logiken (zwischen Modal- und Prädikatenlogik)
  - stochastische Erweiterungen
  - kategoriale Einbettungen
  - ...

- aktuelle Forschungsfragen
  - SW-Verifikation aus UML/OCL, Java-Bytecode
  - Testgenerierung, Synthese von Steuersoftware

III. Softwaretests

- Historisch gewachsen
- viel Technik, wenig Theorie
- Nach wie vor das wichtigste qualitätssichernde Element in jeder SW-Entwicklung
- Zurück zu Dijkstra: "Program testing can be used to show the presence of bugs“
  - Was ist mit Hard- und Middleware?
  - Beweis- und Modellfehler? Eigenschaften?
  - Jeder gefundener Fehler als Erfolg! („erfolgreiche Verifikation“ ist wenig aussagekräftig)

spezifikationsbasierte Tests

- Test nur gegenüber einer Spezifikation sinnvoll
- Marie-Claude Gaudel:
  Testing Can Be Formal, Too (1995)
  - Ableitung von Tests aus Spezifikationen (LOTOS)

- J. Tretmans: Test aus Transitionssystemen (1996)
- Y. Gurevich, W. Grieskamp @Microsoft: ASML, Spec# (2003)

- State of the art: Testgenerierung aus
  - ausführbarem Code (Überdeckungswerkzeuge)
  - Skriptsprachen, z.B. TTCN-3
  - graphischen Modellen, z.B. StateCharts
- Forschungspotential: Testgenerierung aus
  - UML, OCL etc.
  - logischen Spezifikationssprachen
  - natürlicher Sprache
- Studien- und Diplomarbeitsmöglichkeiten!

Zu guter Letzt: Qualitätssicherung...

- … ist nicht nur Spezifizieren, Verifizieren und Testen!
- Prozess-Sicht
  - ISO9000, CMM, SPICE, …
  - FMEA, Review-Techniken, Metriken, …
  - Projektmanagement
  - …
- ➔ Kommen Sie doch in die Vorlesung „Software Engineering II“ im WS 2005/2006!