Einführung

Prof. Sven Apel

Universität des Saarlandes

Basierend auf Material von Andreas Zeller und Norbert Siegmund

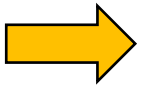
Teil I

Warum Software Engineering?

Warum Software Engineering?

Naive Sicht:

Implementierung

Problemspezifikation  Finales Programm

Aber:

- Woher kommt die *Spezifikation*?
- Wie korrespondiert die Spezifikation zu den *Nutzeranforderungen*?
- Wie entscheidet man, wie das Programm *strukturiert* wird?
- Wie weiß man, dass das Programm *tatsächlich der Spezifikation entspricht*?
- Wie weiß man, dass das Programm immer *korrekt arbeitet*?
- Was macht man, wenn die Nutzeranforderungen *sich ändern*?
- Wie *teilt man Aufgaben auf*, falls man mehr als ein 1-Personen-Team hat?

Softwarekrise

Begriff während der NATO Software Engineering Konferenz 1968 geprägt:

> "[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."
>
> – Edsger Dijkstra, 1972

→ Kosten für Software überstiegen Kosten für Hardware

→ Nur *34%* der Softwareprojekte erfolgreich abgeschlossen

Ariane 5

Ariane 5 Flight 501: 4.6.1996

Selbstzerstörung nach 39 Sekunden

~ $ 500 Million Schaden

However, problems began to occur when the software attempted to *stuff this 64-bit variable*, which can represent billions of potential values, *into a 16-bit integer*, which can only represent 65535 potential values. For the first few seconds of flight, the rocket's acceleration was low, so the conversion between these two values was successful. However, as the rocket's velocity increased, the 64-bit variable exceeded 65k, and became too large to fit in a 16-bit variable.

Therac-25

Medizinische Strahlentherapie

Durch Softwarefehler zu hohe Strahlendosis, 3 Tote

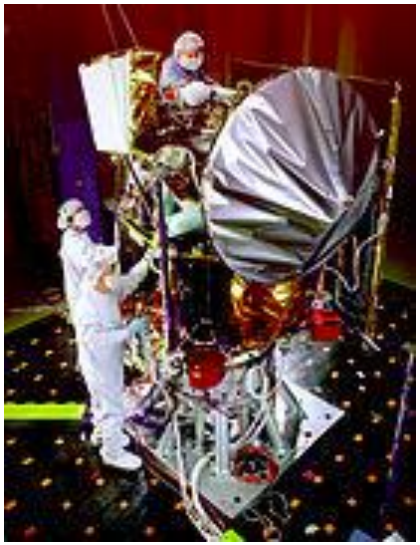
Ein einziger Entwickler schrieb Software und benutzte vorhandene Komponenten

Mars Climate Orbiter

Verloren 1999 beim Anflug auf den Mars

~ $ 330 Millionen Schaden

Grund: Einheitenfehler im Navi
(metrische Einheiten vs. imperiale Einheiten)



Zwischenfall mit Patriot-Rakete

Fehler im Zielsuchsystem führte zum Verfehlen der abzufangenden Rakete und somit zum Tod von 28 Menschen während des Golfkriegs 1991

Grund: Fehlerhafte Zeitberechnung durch ungenaue arithmetische Berechnungen
(Kommastellen wurden bei 24 Bit abgeschnitten, was zu Ungenauigkeiten führte)

AT&T Netzwerkausfall

AT&T-Telefonnetzwerk kollabierte am 15.1.1990

11 h Ausfall, ~ $ 60 Millionen Schaden

"The fault was in the code" of the new software that AT&T loaded into front-end processors of all 114 of its 4ESS switching systems in mid-December, said Larry Seese, AT&T's director of technology development.

Los Angeles Flugkontrolle

Flugkontrolle in LAX stürzte am 30.4.2014 ab

10 gestrichene und 500 verspätete Flüge

Grund: US-Spionageflugzeug verwirrte Software

2003 Northeast Blackout

Über 2 Tage Stromausfall in USA/Kanada

~ 10 Opfer, 45 Millionen Menschen betroffen

Software-Fehler verhinderte lokalen Alarm, so dass sich das Problem ausbreitete

Toyota Motosteuerung

Verursacht “unerwünschte Beschleunigung”

89 Tote zwischen 2001 und 2010

Boeing 737 Max

Jo Atlee @ ESEC/FSE‘19
https://vimeo.com/356373889/dcb19424f9

Absturz Lion Air (189 Tote)

Absturz Ethiopian Airlines (157 Tote)

The angle of attack sensor, at bottom center, is seen on a 737 Max aircraft at the Boeing factory in Renton, Washington, U.S., March 27, 2019. REUTERS/Lindsey Wasson

„The anti-stall system […] has been pinpointed by investigators as a possible cause in a fatal Lion Air crash in Indonesia and the one in Ethiopia.“

Software Fail Watch (5th Edition)

606 erfasste Softwarefehler im Jahr 2017

Betroffen

- 3.7 Milliarden Menschen
- $1.7 Billiarden Kosten
- 314 Unternehmen

Warum scheitern Softwareprojekte?

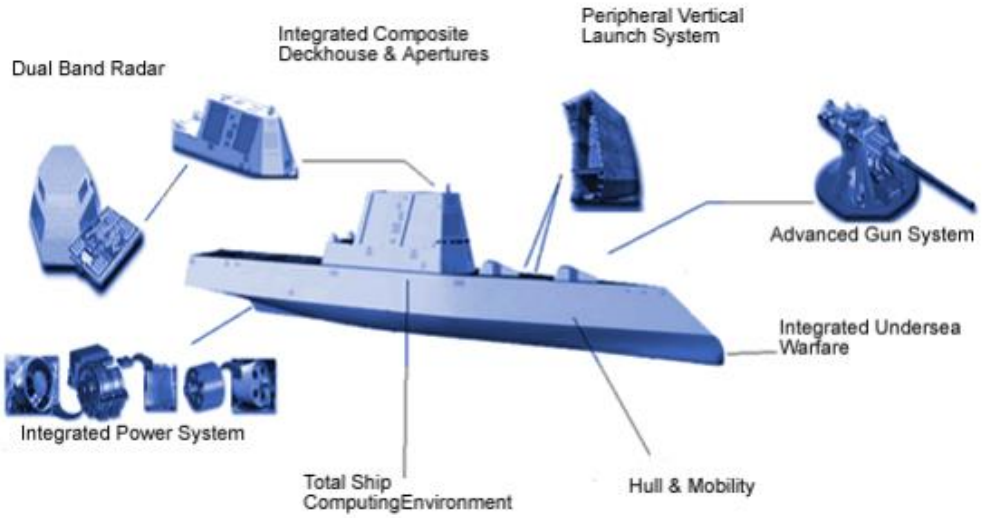
Extreme Komplexität

Beispiel: DDX Zumwalt (US Navy)

Viele, vernetzte eingebettete Systeme

Zusammen 30.000.000.000 Zeilen Code (Schätzung)

in 142 Programmiersprachen



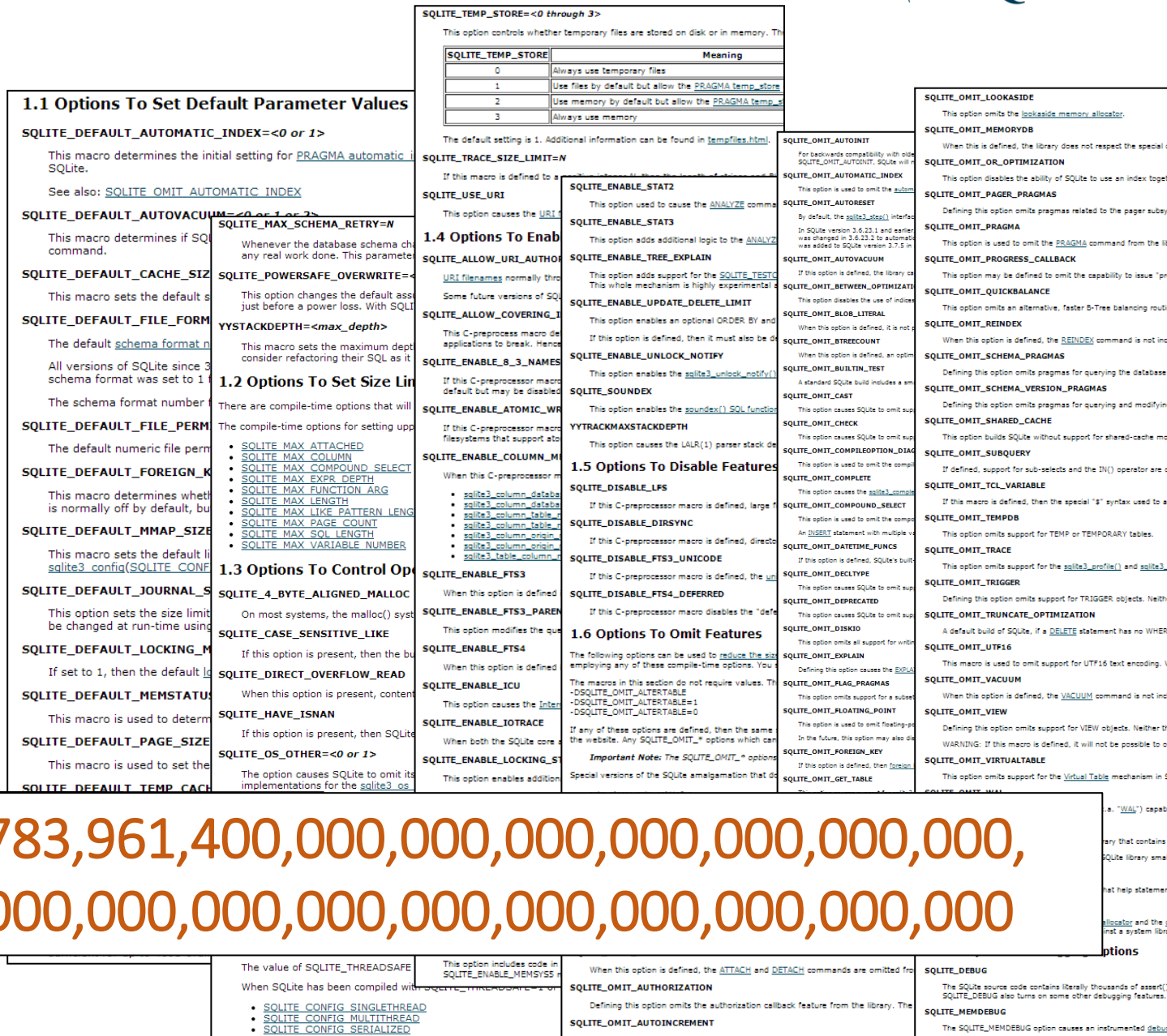
Variabilität

Beispiel: SQLite

- Exclusive Locking
- Case Sensitivity
- Thread Safety
- Atomic Write
- ...

Konfigurationen:

260,532,200,783,961,400,000,000,000,000,000,000,000,
000,000,000,000,000,000,000,000,000,000,000,000,000



Beispiel: SQLite

(88 Konfigurationsoptionen)

Heute

UNIVERSITÄT DES SAARLANDES

13.750.000.000 Jahre

Zeit

9.000.000.000 Jahre

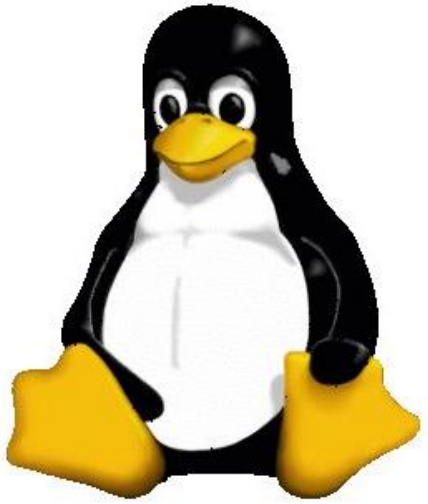
2.900.000.000.000.000.000.000

(2.9×10^{21}) Jahre

Beispiel: SQLite

(88 Konfigurationsoptionen)

Linux-Kernel

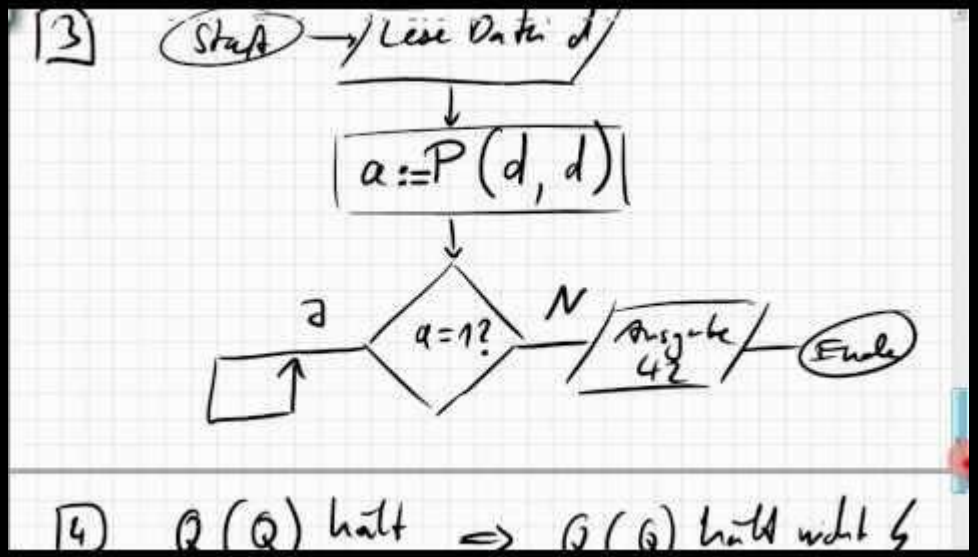


~ *6.000.000* Zeilen Quelltext

Weitgehend konfigurierbar

> *10.000* Konfigurationsoptionen! (x86, 64bit, …)

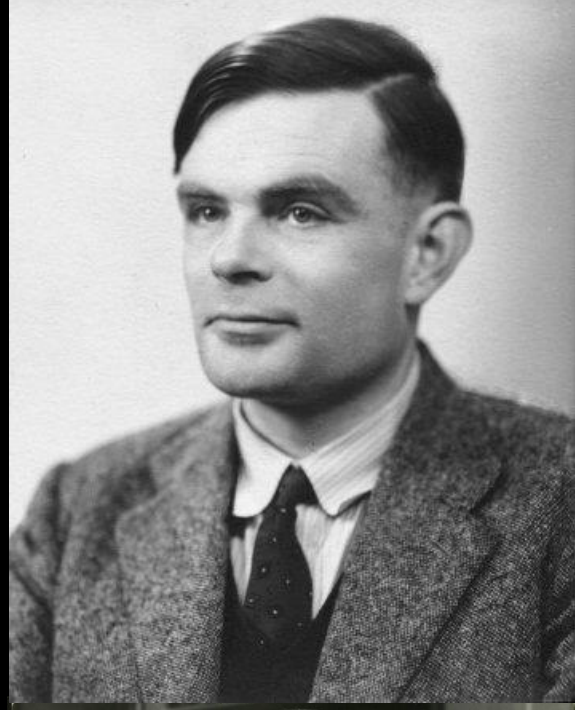
Fast aller Quelltext ist „optional“

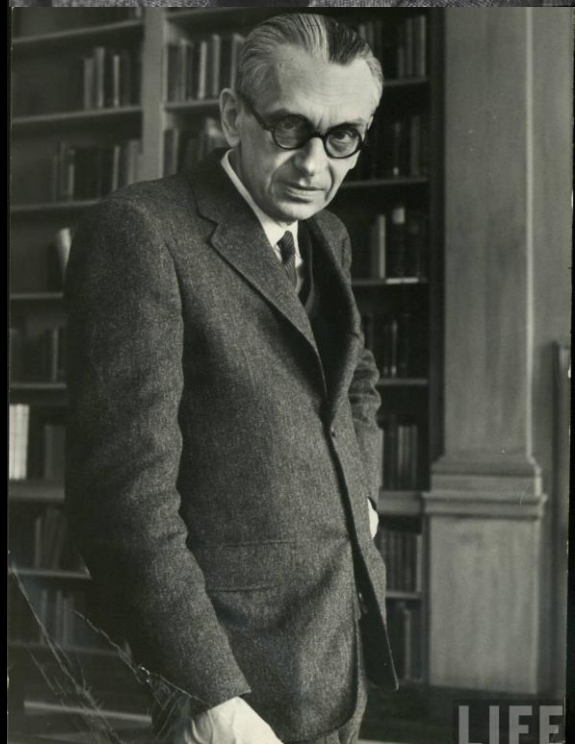


Unentscheidbarkeit

+

Unstetigkeit

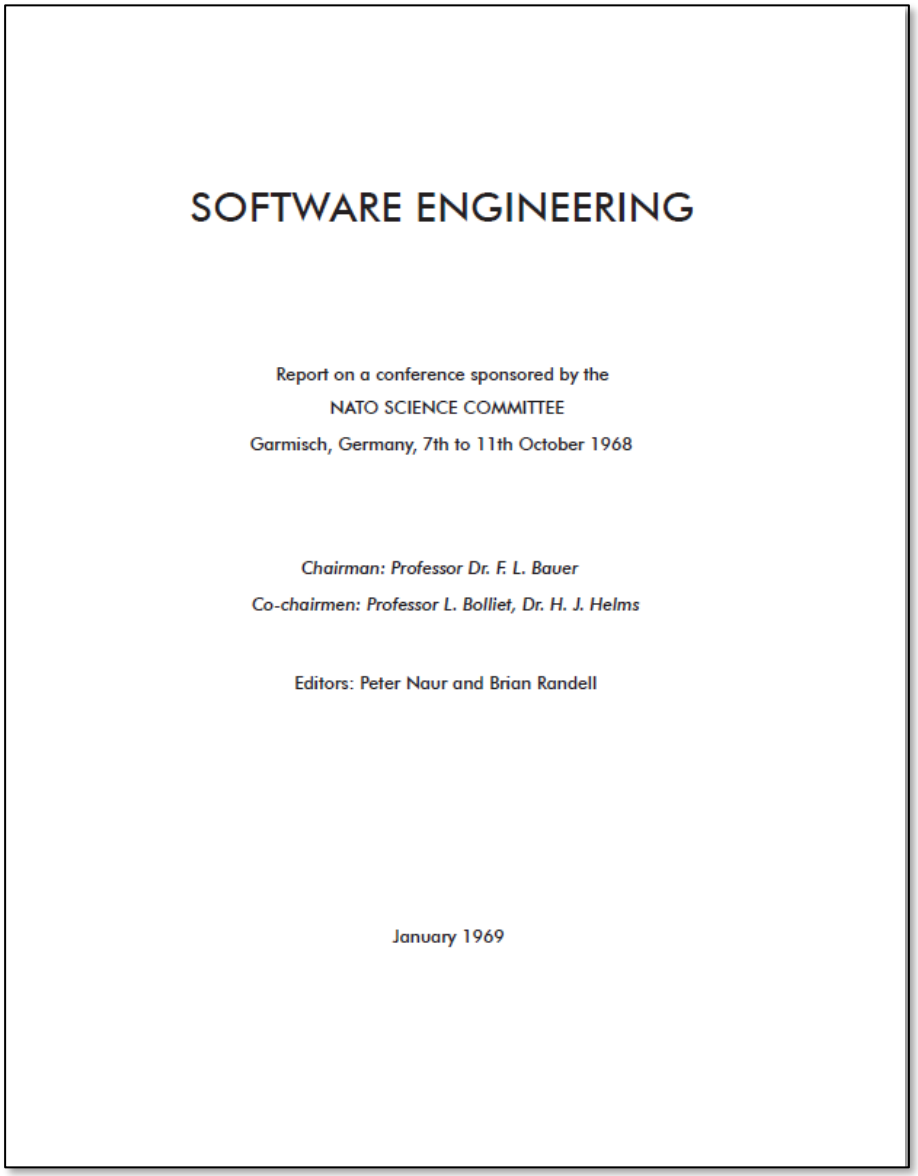




Teil II

Was ist Software Engineering?

Begriff: Software Engineering



SOFTWARE ENGINEERING

Report on a conference sponsored by the
NATO SCIENCE COMMITTEE
Garmisch, Germany, 7th to 11th October 1968

Chairman: Professor Dr. F. L. Bauer
Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms

Editors: Peter Naur and Brian Randell

January 1969

Definition (1)

„*state of the art* of developing *quality* software *on time* and *within budget* “

Aktuellster Stand der Technik:

- Entscheidet Community
- Lebenslanges Lernen

Ressourcenbeschränkung

Definition (2)

„*multi-person* construction of
multi-version software “

Teamwork!

Erfolgreiche Softwaresysteme müssen sich weiterentwickeln

Änderung ist der Normalfall, nicht die Ausnahme

Definition (3)

„software engineering is an *engineering discipline* that is concerned with *all aspects* of *software production*“

Software ist ein *Produkt*, das für einen bestimmtem Kunden entwickelt wird

Nicht nur Programmierung, sondern *alle Aspekte des Softwarelebenszyklus*

Software Engineering vs. Informatik

Informatik beschäftigt sich mit der Theorie und den Grundlagen von Computersystemen

Software Engineering beschäftigt sich mit theoretischen und praktischen Themen der Entwicklung und Wartung “guter” Software

Was ist „gute“ Software?

Maintainable (Wartbar)

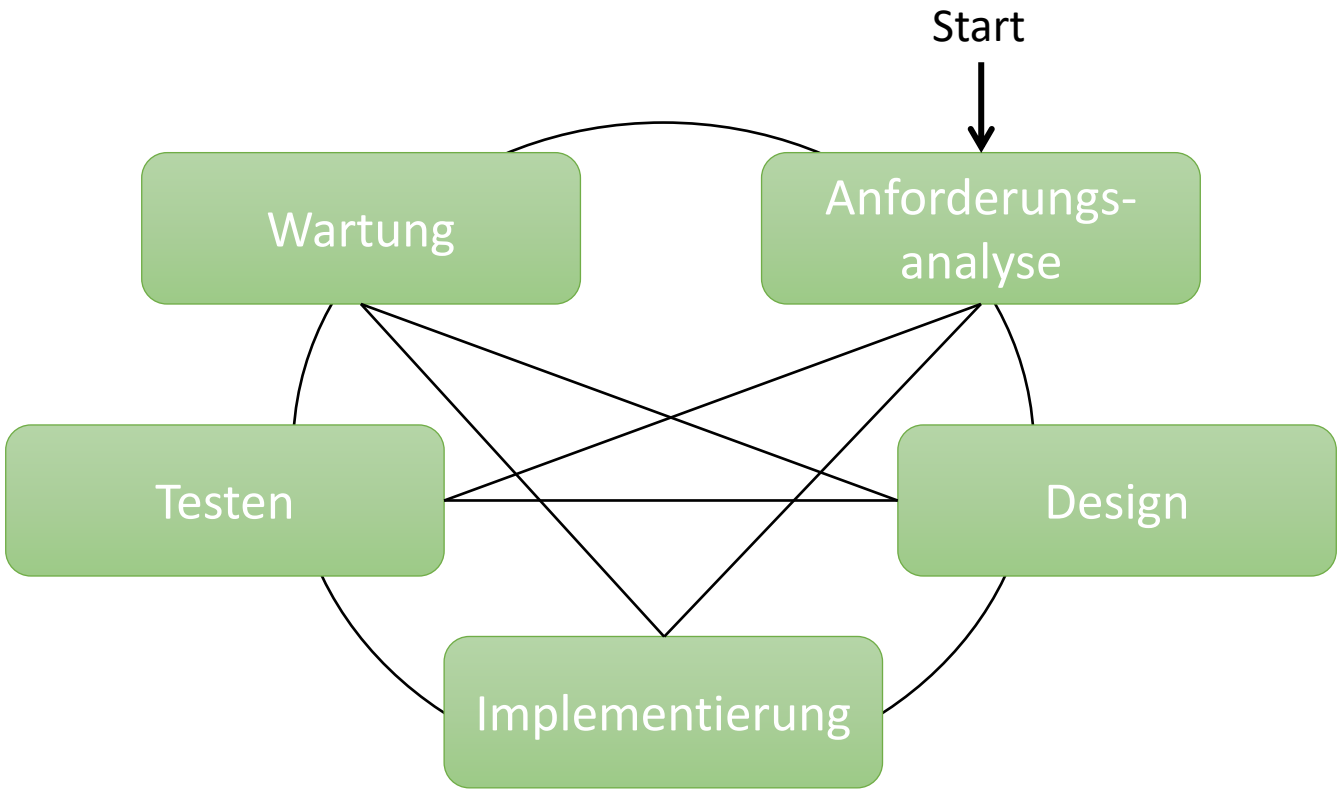
Dependable (Verfügbar, Zuverlässig)

Efficient (Effizient)

Usable (Nutzbar, Anwendbar)

Secure, Safe (Sicher)

Phasen Softwarelebenszyklus



Programmierung ist nur ein kleiner Teil von Softwareentwicklung!

Abgrenzung: Ingenieursdisziplin

Software hat praktisch *keine physischen* Eigenschaften

Keine zugrundeliegende physikalische Theorie

Kein Verfall, trotzdem existiert nach 10 Jahren typischerweise keine einzige Zeile des ursprünglichen Quellcodes mehr

Software kann (fast) *beliebig komplex* werden