Fakultät Informatik/Mathematik

HOCHSCHULE FÜR
TECHNIK UND WIRTSCHAFT
DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Vorlesungsmitschrift

SPEZIELLE THEMEN/TECHNOLOGIEN DER INFORMATIK

Mitschrift von

Falk-Jonatan Strube

Vorlesung von

Prof. Dr.-Ing. habil. Peter Sobe
Prof. Dr. rer. pol. Dirk Reichelt
Prof. Dr. rer. nat. Marco Block-Berlitz
Prof. Dr.-Ing. Robert Baumgartl
Prof. Dr.-Ing. Anna Sabine Hauptmann

27. März 2018

INHALTSVERZEICHNIS

1 Reliability of Computer Systems **6**

1.1 Einführung . . . 6
1.1.1 Zuverlässige / fehlertolerante Systeme . . . 6
1.1.2 Modellierung fehlertoleranter Systeme . . . 6
1.2 Grundlegende Zuverlässigkeitsquantifizierung . . . 6
1.2.1 Wie lang läuft ein System ohne Versagen? . . . 7
1.2.2 Wie Wahrscheinlich ist die fehlerfreie Nutzung für einen bestimmten Zeitraum? . . . 7
1.2.3 Versagenswahrscheinlichkeit und Zuverlässigkeit . . . 7
1.2.4 Mittlere Zeit bis zum Versagen (MTTF) . . . 7
1.2.5 Verfügbarkeit . . . 8
1.2.6 Beispiel Versagen und Verfügbarkeit . . . 8
1.2.6.1 Einzelne Komponente ohne Reparatur . . . 8
1.2.6.2 Einzelne Komponente mit Reparatur . . . 8
1.2.7 Systemstruktur . . . 8
1.2.7.1 Zusammensetzung in Reihe . . . 8
1.2.7.2 Parallele Zusammensetzung . . . 9
1.2.8 Zusammenfassung . . . 9
1.3 Zuverlässige/Fehlertolerante Systeme . . . 9
1.3.1 Verlässliches System . . . 9
1.3.2 Fehlertolerantens System . . . 9
1.3.3 Fehlerklassen . . . 10
1.3.4 Techniken zur Fehlertoleranz . . . 10
1.3.4.1 Redundanz . . . 10
1.3.4.2 Fehlerisolation, Wiederherstellung und Fehlerkompensation . . . 10
1.3.4.3 Rekonfigurierung (DMR, PSR, TMPR) . . . 10
1.3.5 Systemschichten und Fehlertoleranztechniken . . . 11
1.3.6 Grundlegende Techniken der Fehlertoleranz . . . 11
1.3.7 Zusammenfassung . . . 11
1.4 Zuverlässigkeitsmodellierung . . . 11
1.4.1 Systemstruktur . . . 12
1.4.2 Boolsche Modelle . . . 12
1.4.3 Veranschaulichung Boolscher Modelle . . . 12
1.4.3.1 Fehlerbäume . . . 12
1.4.3.2 RBD (Reliability Block Diagramm) . . . 12
1.4.4 Arbeiten mit Wahrscheinlichkeiten . . . 12
1.4.4.1 (k von k+m)-System . . . 12
1.4.4.2 R und F als Funktionen der Zeit . . . 13
1.4.5 Zussamenfassung: Statische Systemkonfigurierung . . . 13
1.4.6 Dynamische fehlertolerante Systeme mit Reparatur . . . 13
1.4.7 Markov Modelle . . . 13
1.4.7.1 Einzelne Komponente ohne Reparatur . . . 13
1.4.7.2 Einzelne Komponente mit Reparatur . . . 13
1.4.7.3 Allgemeines Markov Modell eines Zweikomponentensystems . . . 14
1.4.7.4 Allgemeines Markov Modell eines Dreikomponentensystems . . . 14

1.4.7.5 Markovketten Simulator . . . 14
1.4.8 Zusammenfassung . . . 14

2 Semantische Interoperabilität für M2M Szenarien / Genetische Algorithmen 15
2.1 Einführung in IoT-Protokolle . . . 15
2.1.1 Das Szenario . . . 15
2.1.2 Schaltzentrale IoT-Hub . . . 15
2.1.2.1 Dashboards . . . 15
2.1.2.2 Einfache Regeln . . . 15
2.1.3 Business Rules via iLog . . . 15
2.1.4 IoT-Baukasten – Node RED . . . 15
2.2 Enterprise Integration Pattern . . . 16
2.2.1 Anwendungsintegration . . . 16
2.2.1.1 Point-to-Point (P2P) . . . 16
2.2.1.2 Message Bus . . . 16
2.2.1.3 Publish-Subscribe . . . 16
2.2.1.4 P2P versus Publish-Subscribe . . . 16
2.2.1.5 Durable Subscriber . . . 16
2.3 Messaging mit Java = JMS . . . 17
2.4 MQTT – sprachunabhängig und leichtgewichtig . . . 17
2.4.1 Features . . . 17
2.4.2 Quality of Service . . . 17
2.4.3 Populäre Broker und Clients . . . 17
2.4.4 Java Anwendung . . . 17
2.4.4.1 Publisher . . . 17
2.4.4.2 Subscriber . . . 17
2.5 Aufgaben . . . 18
2.6 REST und SOAP . . . 18

3 3D-Rekonstruktion 19

4 Reverse Engineering 20
4.1 Einführung . . . 20
4.1.1 Zweck . . . 20
4.1.2 Rechtliche Aspekte . . . 20
4.1.3 Beschränkung . . . 20
4.2 Werkzeuge . . . 20
4.3 Assembler . . . 20
4.3.1 IA-32 CPU . . . 20
4.3.1.1 Wichtige Register . . . 20
4.3.1.2 EFLAGS . . . 21
4.3.2 Instruktionen . . . 21
4.3.3 Stack . . . 21
4.3.3.1 push und pop . . . 22
4.3.4 Calling Conventions . . . 22
4.4 Konstrukte in C . . . 22
4.4.1 if-else-Statement . . . 22
4.4.2 for-Schleife . . . 24
4.4.2.1 goto-Statement . . . 24
4.4.3 Funktionsaufrufe . . . 24
4.4.3.1 Parameter- und Ergebniskommunikation . . . 25
4.4.3.2 Struktur . . . 25

4.4.4 Systemrufe 25
4.5 gdb 26

5 Antipattern im Kontext der Software-Entwicklung 27
5.1 Muster vs Antimuster 27
5.1.1 Muster 27
5.1.2 Anti-Muster 27
5.1.3 Referenz-Modelle 27
5.2 Refactoring 28
5.3 Anti-Muster 29
5.3.1 The Blob 29
5.3.2 Lava Flow 29
5.3.3 Poltergeister 30
5.3.4 Goldener Hammer 30
5.3.5 Spaghetti-Code 30
5.4 Sotograph 32
5.5 Antimuster im Kontext der Architektur 32
5.5.1 Stovepipe, autogenerated 33
5.5.2 Stovepipe Enterprise 33

VORBEMERKUNGEN

BELEGE

Teil-Beleg Ausgabe von jedem Professor.
3-aus-5 Auswahl der besten Teilergebnisse.

1 RELIABILITY OF COMPUTER SYSTEMS

Vorlesungen von: Prof. Dr.-Ing. habil. Peter Sobe

Vorlesung
20.03.2017

STTI_intro_2017.pdf
Folie 6

1.1 EINFÜHRUNG

ZIEL DER VORLESUNG

reliability/contents.pdf
Folie 2

1.1.1 ZUVERLÄSSIGE / FEHLERTOLERANTE SYSTEME

reliability/contents.pdf
Folie 4

Einfachstes Mittel für besser Reliability: REDUNDANZ
Failur classes . . . meint Fehler-Klassen von kritischen Fehlern (Versagen)

1.1.2 MODELLIERUNG FEHLERTOLERANTER SYSTEME

reliability/contents.pdf
Folie 5

1.2 GRUNDLEGENDE ZUVERLÄSSIGKEITSQUANTIFIZIERUNG

reliability/contents.pdf
Folie 3

Reliability . . . Zuverlässigkeit / Wahrscheinlichkeit, dass ein System über eine Zeit funktionsfähig ist
Availability . . . Verfügbarkeit (pro Zeit)
Mission time . . . Wie lange soll ein System ohne Reparatur „im Feld“ bleiben?

1.2.1 WIE LANG LÄUFT EIN SYSTEM OHNE VERSAGEN?

reliability/part1.pdf
Folie 3

reliability/part1.pdf
Folie 4

Also: Der Mittelwert gibt keine Garantie, dass das Gerät bis zu einer bestimmten Zeit läuft.

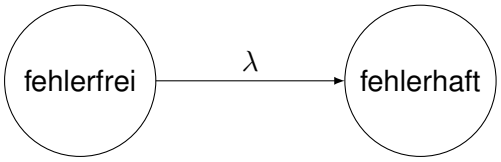
1.2.2 WIE WAHRSCHEINLICH IST DIE FEHLERFREIE NUTZUNG FÜR EINEN BESTIMMTEN ZEITRAUM?

reliability/part1.pdf
Folie 5

reliability/part1.pdf
Folie 6

1.2.3 VERSAGENSWAHRSCHEINLICHKEIT UND ZUVERLÄSSIGKEIT

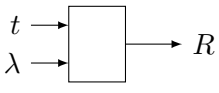
reliability/part1.pdf
Folie 7



$R(t) = P(fehlerfrei)$
$R(t) = e^{-\lambda t}$
$R'(t) = -\lambda R(t)$

1.2.4 MITTLERE ZEIT BIS ZUM VERSAGEN (MTTF)

reliability/part1.pdf
Folie 8



$MTTF = P(L > t) = E(t) = \frac{1}{\lambda}$
Bspw.: $\lambda = 5\frac{1}{year} \rightarrow MTTF = \frac{1}{5}year$

1.2.5 VERFÜGBARKEIT

reliability/part1.pdf
Folie 9

reliability/part1.pdf
Folie 10

1.2.6 BEISPIEL VERSAGEN UND VERFÜGBARKEIT

1.2.6.1 EINZELNE KOMPONENTE OHNE REPARATUR

reliability/part1.pdf
Folie 11

Die Einsatzdauer sollte immer deutlich kürzer sein, als die MTTF (da $R(MTTF) = 0,37$ – also die Fehlerrate zur mittleren Versagenszeit – ein sehr schlechter Wert ist).

1.2.6.2 EINZELNE KOMPONENTE MIT REPARATUR

reliability/part1.pdf
Folie 12

1.2.7 SYSTEMSTRUKTUR

reliability/part1.pdf
Folie 13

1.2.7.1 ZUSAMMENSETZUNG IN REIHE

Vorlesung
27.03.2017

reliability/part1.pdf
Folie 14

reliability/part1.pdf
Folie 15

VERFÜGBARKEIT

reliability/part1.pdf
Folie 16

1.2.7.2 PARALLELE ZUSAMMENSETZUNG

reliability/part1.pdf
Folie 17

reliability/part1.pdf
Folie 18

1.2.8 ZUSAMMENFASSUNG

reliability/part1.pdf
Folie 20

reliability/part1.pdf
Folie 21

Beispiele für Systeme, die zuverlässig sein müssen: Flugzeug(-klappensteuerung), Atomkraftwerksteuerung, medizinische Geräte, Rechenzentren, ...

1.3 ZUVERLÄSSIGE/FEHLERTOLERANTE SYSTEME

reliability/part2.pdf
Folie 2

1.3.1 VERLÄSSLICHES SYSTEM

reliability/part2.pdf
Folie 3

reliability/part2.pdf
Folie 4

1.3.2 FEHLERTOLERANTENS SYSTEM

reliability/part2.pdf
Folie 5

reliability/part2.pdf
Folie 6

1.3.3 FEHLERKLASSEN

reliability/part2.pdf
Folie 7

(von unten noch oben immer allgemeinere Fehler)

reliability/part2.pdf
Folie 8

1.3.4 TECHNIKEN ZUR FEHLERTOLERANZ

1.3.4.1 REDUNDANZ

reliability/part2.pdf
Folie 9

reliability/part2.pdf
Folie 10

reliability/part2.pdf
Folie 11

1.3.4.2 FEHLERISOLATION, WIEDERHERSTELLUNG UND FEHLERKOMPENSATION

reliability/part2.pdf
Folie 12

1.3.4.3 REKONFIGURIERUNG (DMR, PSR, TMPR)

reliability/part2.pdf
Folie 13

DMR SYSTEM

reliability/part2.pdf
Folie 14

PSR SYSTEM

reliability/part2.pdf
Folie 15

TMPR SYSTEM

reliability/part2.pdf
Folie 16

REPLIKATION MIT VERTEILTER MEHRHEITSWAHL

reliability/part2.pdf
Folie 17

1.3.5 SYSTEMSCHICHTEN UND FEHLERTOLERANZTECHNIKEN

reliability/part2.pdf
Folie 19

reliability/part2.pdf
Folie 20

reliability/part2.pdf
Folie 21

1.3.6 GRUNDLEGENDE TECHNIKEN DER FEHLERTOLERANZ

Vorlesung
03.04.2017

reliability/part2.pdf
Folie 22

BEISPIELE

reliability/part2.pdf
Folie 23

reliability/part2.pdf
Folie 24

1.3.7 ZUSAMMENFASSUNG

reliability/part2.pdf
Folie 25

1.4 ZUVERLÄSSIGKEITSMODELLIERUNG

reliability/part3.pdf
Folie 2

1.4.1 SYSTEMSTRUKTUR

reliability/part3.pdf
Folie 3

1.4.2 BOOLSCHE MODELLE

reliability/part3.pdf
Folie 4

1.4.3 VERANSCHAULICHUNG BOOLSCHER MODELLE

reliability/part3.pdf
Folie 5

1.4.3.1 FEHLERBÄUME

reliability/part3.pdf
Folie 6

BEISPIELE

reliability/part3.pdf
Folie 7

reliability/part3.pdf
Folie 8

1.4.3.2 RBD (RELIABILITY BLOCK DIAGRAMM)

reliability/part3.pdf
Folie 9

1.4.4 ARBEITEN MIT WAHRSCHEINLICHKEITEN

reliability/part3.pdf
Folie 10

1.4.4.1 (K VON K+M)-SYSTEM

reliability/part3.pdf
Folie 11

BEISPIELE

reliability/part3.pdf
Folie 12

reliability/part3.pdf
Folie 13

1.4.4.2 R UND F ALS FUNKTIONEN DER ZEIT

reliability/part3.pdf
Folie 14

reliability/part3.pdf
Folie 15

1.4.5 ZUSSAMENFASSUNG: STATISCHE SYSTEMKONFIGURIERUNG

reliability/part3.pdf
Folie 16

1.4.6 DYNAMISCHE FEHLERTOLERANTE SYSTEME MIT REPARATUR

reliability/part3.pdf
Folie 17

1.4.7 MARKOV MODELLE

reliability/part3.pdf
Folie 21

reliability/part3.pdf
Folie 22

1.4.7.1 EINZELNE KOMPONENTE OHNE REPARATUR

reliability/part3.pdf
Folie 23

1.4.7.2 EINZELNE KOMPONENTE MIT REPARATUR

reliability/part3.pdf
Folie 24

1.4.7.3 ALLGEMEINES MARKOV MODELL EINES ZWEIKOMPONENTENSYSTEMS

reliability/part3.pdf
Folie 25

1 VON 2 SYSTEM MIT REPARATUR

reliability/part3.pdf
Folie 26

1.4.7.4 ALLGEMEINES MARKOV MODELL EINES DREIKOMPONENTENSYSTEMS

reliability/part3.pdf
Folie 27

3 VON 2 SYSTEM OHNE REPARATUR

reliability/part3.pdf
Folie 28

2 VON 3 SYSTEM MIT REPERATUR

reliability/part3.pdf
Folie 29

1.4.7.5 MARKOVKETTEN SIMULATOR

reliability/part3.pdf
Folie 30

1.4.8 ZUSAMMENFASSUNG

reliability/part3.pdf
Folie 31

reliability/part3.pdf
Folie 32

2 SEMANTISCHE INTEROPERABILITÄT FÜR M2M SZENARIEN / GENETISCHE ALGORITHMEN

Vorlesungen von: Prof. Dr. rer. pol. Dirk Reichelt

2.1 EINFÜHRUNG IN IOT-PROTOKOLLE

Vorlesung
10.04.2017

2.1.1 DAS SZENARIO

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 2

2.1.2 SCHALTZENTRALE IOT-HUB

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 3

2.1.2.1 DASHBOARDS

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 4

2.1.2.2 EINFACHE REGELN

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 5

2.1.3 BUSINESS RULES VIA ILOG

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 6

2.1.4 IOT-BAUKASTEN – NODE RED

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 7

2.2 ENTERPRISE INTEGRATION PATTERN

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 8

Übertragungsoptionen:

- Über Verbindung (bspw. http)
  Nachteil: beide Gerät müssen an sein für Kommunkation
- Über Datei
  Nachteil: Struktur der Datei muss bekannt sein
  Vorteil: beide Geräte müssen nicht an sein (bspw. über USB-Stick)
- Über Datenbank
  Nachteil: zusätzlicher Verwaltungsaufwand nötig
  Vorteil: klare Struktur

2.2.1 ANWENDUNGSINTEGRATION

2.2.1.1 POINT-TO-POINT (P2P)

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 9

2.2.1.2 MESSAGE BUS

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 10

2.2.1.3 PUBLISH-SUBSCRIBE

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 11

2.2.1.4 P2P VERSUS PUBLISH-SUBSCRIBE

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 12

2.2.1.5 DURABLE SUBSCRIBER

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 13

(Kombination aus Publish-Subscribe und P2P)

2.3 MESSAGING MIT JAVA = JMS

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 14

2.4 MQTT – SPRACHUNABHÄNGIG UND LEICHTGEWICHTIG

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 15

(MQTT: Message Queue Telemetry Transport)

2.4.1 FEATURES

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 16

2.4.2 QUALITY OF SERVICE

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 17

(QoS: Quality of Service)

2.4.3 POPULÄRE BROKER UND CLIENTS

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 18

2.4.4 JAVA ANWENDUNG

2.4.4.1 PUBLISHER

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 20

2.4.4.2 SUBSCRIBER

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 21

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 22

2.5 AUFGABEN

IoT/STTI_SoSe2017_CloudMQTT.pdf
Folie 25

1. Siehe Java-files
2. Siehe Java-files
3. Siehe Java-files
4. - QoS 0: Temperatur (nicht schlimm, wenn eine Nachricht verloren geht)
   - QoS 1: Stückzahl Zähler (Mehrfachempfang von Nachricht ist nicht relevant)
   - QoS 2: Überweisung (Nachricht muss GENAU ein Mal empfangen werden)
5. „Persistent Sessions“ setzt das „Durable Subscriber Pattern“ um:
   Bei häufigen Verbindungsabbrüchen von Clients kann der Broker eine persistente Session vorhalten. Wenn ein Client sich erneut verbindet schickt der Broker alle verpassten Nachrichten für seine Subscriptions an den Client.
   Dafür muss `cleanSessions` auf `false` gesetzt werden.

2.6 REST UND SOAP

Vorlesung
24.04.2017

...

3 3D-REKONSTRUKTION

Vorlesungen von: Prof. Dr. rer. nat. Marco Block-Berlitz

STTI_intro_2017.pdf
Folie 8

→ Praktika

Vorlesung
08.05.2017

4 REVERSE ENGINEERING

Vorlesungen von: Prof. Dr.-Ing. Robert Baumgartl

4.1 EINFÜHRUNG

reverseEngineering/stti-01-einfuehrung.pdf
Folie 2

4.1.1 ZWECK

reverseEngineering/stti-01-einfuehrung.pdf
Folie 4

4.1.2 RECHTLICHE ASPEKTE

reverseEngineering/stti-01-einfuehrung.pdf
Folie 5

4.1.3 BESCHRÄNKUNG

reverseEngineering/stti-01-einfuehrung.pdf
Folie 6

4.2 WERKZEUGE

reverseEngineering/stti-01-einfuehrung.pdf
Folie 7

4.3 ASSEMBLER

4.3.1 IA-32 CPU

4.3.1.1 WICHTIGE REGISTER

reverseEngineering/stti-01-einfuehrung.pdf
Folie 9

reverseEngineering/stti-01-einfuehrung.pdf
Folie 10

E: Vorsatz für 32-bit Prozessoren (R für 64-bit).
A-DX: Universalregister
SI: Source Index
DI: Destination Index
BP: Base Pointer
SP: Stack Pointer (TOS: Top of Stack)
IP: Instruction Pointer
FLAGS: Flags (Zeigen Situationen an [wenn bspw. 0 aus Operation kommt: zero-flag])

4.3.1.2 EFLAGS

reverseEngineering/stti-01-einfuehrung.pdf
Folie 11

4.3.2 INSTRUKTIONEN

reverseEngineering/stti-01-einfuehrung.pdf
Folie 12

- `mov`: kopiere Wert von A nach B (nicht bewegen/verschieben).
- `mul`: Multiplizieren (`mul %eax, %ebx`$\rightarrow ebx = ebx \cdot eax$)
- `add`: Addieren
- `sub`: Subtrahieren
- `inc`: Inkrementieren
- `dec`: Dekrementieren
- `push`: Schreibe in Stack
- `pop`: Hole vom Stack
- `call`: Rufe Programm auf (und merke, wo man war)
- `ret`: springt zurück wo man vorher war (nach `call`)
- `jmp`: Sprung
- `jcc`: Sprung mit Condition Code (siehe Folie 11)
- `cmp`: Vergleiche
- `test`:

4.3.3 STACK

reverseEngineering/stti-01-einfuehrung.pdf
Folie 13

STACKFRAME

reverseEngineering/stti-01-einfuehrung.pdf
Folie 14

4.3.3.1 PUSH UND POP

reverseEngineering/stti-01-einfuehrung.pdf
Folie 15

AUFBAU

reverseEngineering/stti-01-einfuehrung.pdf
Folie 16

4.3.4 CALLING CONVENTIONS

reverseEngineering/stti-01-einfuehrung.pdf
Folie 17

4.4 KONSTRUKTE IN C

reverseEngineering/stti-01-einfuehrung.pdf
Folie 18

4.4.1 IF-ELSE-STATEMENT

reverseEngineering/stti-01-einfuehrung.pdf
Folie 19

Ausführen: `gdb applicationName`
Dann: `disas main`
Alternativ: `objdump -d applicationName`

```
0x080483fb  <+0>: lea 0x4(%esp),%ecx
0x080483ff  <+4>: and $0xfffffff0,%esp
0x08048402  <+7>: pushl -0x4(%ecx)
0x08048405 <+10>: push %ebp
; Schreibe Stack-Pointer in Base-Pointer:
0x08048406 <+11>: mov %esp,%ebp
0x08048408 <+13>: push %ecx
; Subtrahiere 20 von ESP (Platz für lokale Variablen wird
   reserviert):
0x08048409 <+14>: sub $0x14,%esp
; Initialisiere Variablen; Schreibe 1 in EBP-12 (Variable b):
0x0804840c <+17>: movl $0x1,-0xc(%ebp)
; Vergleiche 42 mit b:
```

```
0x08048413 <+24>: cmpl $0x2a,-0xc(%ebp)
; (Über-)springe, wenn 42!=1:
0x08048417 <+28>: jne 0x8048422 <main+39>
; Schreibe 23 in EBP-16 (Variable a):
0x08048419 <+30>: movl $0x17,-0x10(%ebp)
; (Über-)springe:
0x08048420 <+37>: jmp 0x8048429 <main+46>
; Schreibe 42 in ESP-16 (Variable a):
0x08048422 <+39>: movl $0x2a,-0x10(%ebp)
; Vergrößere Etack (nicht nötig?):
0x08048429 <+46>: sub $0xc,%esp
; Ende des Stacks auf 0 setzen (als Übergabeparameter für exit):
0x0804842c <+49>: push $0x0
; Rufe exit
0x0804842e <+51>: call 0x80482e0 <exit@plt>
```

```
0x0804842b  <+0>: lea 0x4(%esp),%ecx
0x0804842f  <+4>: and $0xfffffff0,%esp
0x08048432  <+7>: pushl -0x4(%ecx)
0x08048435 <+10>: push %ebp
0x08048436 <+11>: mov %esp,%ebp
0x08048438 <+13>: push %ecx
0x08048439 <+14>: sub $0x14,%esp
; Setze c=0 (Hinweis: negativer Offset im lokale Variable):
0x0804843c <+17>: movl $0x0,-0xc(%ebp)
; Springe zu main+49:
0x08048443 <+24>: jmp 0x804845c <main+49>
; Erweitere Stack um 8 Bit:
0x08048445 <+26>: sub $0x8,%esp
; Schreibe Wert der Variablen auf Stack:
0x08048448 <+29>: pushl -0xc(%ebp)
; Schreibe konstante Adresse (Formatstring für printf):
0x0804844b <+32>: push $0x8048500
; Rufe printf auf:
0x08048450 <+37>: call 0x80482f0 <printf@plt>
; Verkleinere Stack wieder um 16 Bit (8 erzeugte und 8 gepushte):
0x08048455 <+42>: add $0x10,%esp
; Addiere 1 auf b:
0x08048458 <+45>: addl $0x1,-0xc(%ebp)
; Vergleiche mit 23:
0x0804845c <+49>: cmpl $0x16,-0xc(%ebp)
; Wenn kleiner/gleich 22 (also kleiner 23), springe zurück zu main
   +26.
0x08048460 <+53>: jle 0x8048445 <main+26>
0x08048462 <+55>: sub $0xc,%esp
; Wieder 0 auf Stack pushen...
0x08048465 <+58>: push $0x0
; ...und exit(0):
0x08048467 <+60>: call 0x8048310 <exit@plt>
```


reverseEngineering/stti-01-einfuehrung.pdf
Folie 21

4.4.2 FOR-SCHLEIFE

reverseEngineering/stti-01-einfuehrung.pdf
Folie 22

reverseEngineering/stti-01-einfuehrung.pdf
Folie 23

reverseEngineering/stti-01-einfuehrung.pdf
Folie 24

4.4.2.1 GOTO-STATEMENT

reverseEngineering/stti-01-einfuehrung.pdf
Folie 25

4.4.3 FUNKTIONSAUFRUFE

reverseEngineering/stti-01-einfuehrung.pdf
Folie 26

```
(gdb) disas main
...
0x0804844b <+11>: mov %esp,%ebp ; neuer Stackframe
0x0804844d <+13>: push %ecx
0x0804844e <+14>: sub $0x14,%esp ; Platz fuer lokale Variablen
0x08048451 <+17>: movl $0x17,-0xc(%ebp) ; a=23
0x08048458 <+24>: movl $0x2a,-0x10(%ebp) ; b=42
0x0804845f <+31>: pushl -0x10(%ebp) ; push b
0x08048462 <+34>: pushl -0xc(%ebp) ; push a
0x08048465 <+37>: call 0x804842b <mul>
0x0804846a <+42>: add $0x8,%esp ; Parameter vom Stack
0x0804846d <+45>: mov %eax,-0x14(%ebp) ; erg=mul(a,b);
0x08048470 <+48>: sub $0x8,%esp
0x08048473 <+51>: pushl -0x14(%ebp) ; push erg
0x08048476 <+54>: push $0x8048520 ; push &"Ergebnis: %d\n"
0x0804847b <+59>: call 0x80482f0 <printf@plt>
0x08048480 <+64>: add $0x10,%esp ; Parameter vom Stack
0x08048483 <+67>: sub $0xc,%esp
0x08048486 <+70>: push $0x0 ; push 0
0x08048488 <+72>: call 0x8048310 <exit@plt> ; exit(0)
(gdb) disas mul
0x0804842b  <+0>: push %ebp ; akt. Stackframe sichern
```

```
0x0804842c   <+1>: mov %esp,%ebp ; neuer Stackframe
0x0804842e   <+3>: sub $0x10,%esp ; Platz fuer lokale Variablen
0x08048431   <+6>: mov 0x8(%ebp),%eax ; eax:=a (Parameter)
0x08048434   <+9>: imul 0xc(%ebp),%eax ; eax:=a*b
0x08048438 <+13>: mov %eax,-0x4(%ebp) ; result=a*b
0x0804843b <+16>: mov -0x4(%ebp),%eax ; Resultat in eax
0x0804843e <+19>: leave ; esp:=ebp; pop ebp
0x0804843f <+20>: ret ; pop eip
```

4.4.3.1 PARAMETER- UND ERGEBNISKOMMUNIKATION

reverseEngineering/stti-01-einfuehrung.pdf
Folie 28

4.4.3.2 STRUKTUR

reverseEngineering/stti-01-einfuehrung.pdf
Folie 29

4.4.4 SYSTEMRUFE

reverseEngineering/stti-01-einfuehrung.pdf
Folie 30

AUFRUFKONVENTION SYSTEMRUF LINUX

reverseEngineering/stti-01-einfuehrung.pdf
Folie 31

```
.text
.global _start
_start:
movl $0, %eax
xorl %ebx, %ebx ; setze ebx und edx=0 .. xor macht es sicherer als
   es direkt auf 0 zu setzen!
xorl %edx, %edx
jmp string
code:
pop %ecx  ; Speicher Adresse des Strings in %ecx (%c: addr von
   write)
movb $01, %bl ; stdout (%b: filedesc von write)
movb $15, %dl ; string (%d: lgth von write)
movb $04, %al ; 'write(stdout, addr, lgth)'
int $0x80
decb %bl
movb $01,%al  ; 'exit(0)'
int $0x80
```

```
17 string:
18 call code ; call ermittelt die Adresse des folgenden Strings durch
     das implizite push
19 .ascii "Hello, world!\x0a\x00"
```

Äquivalent dazu das C-Programm:

```
1 int man(void) {
2   printf("Hello, world!\n");
3   exit 0;
4 }
```

4.5 GDB

gdb Kommandos:

- `disas Funktion`
- `br Adresse` (Breakpoint) bps.: `br *0x08048b4`
- `run` (läuft [bis zum Breakpoint])
- `cont` (setzt [nach Breakpoint] fort)
- `inf reg` (Informationen über vorhandene Register)
- `x/bx 0xbffff638` (Information zu Inhalt von Register[→durch `inf reg` in Erfahrung bringen], mit Enter die folgenden Adressen)

5 ANTIPATTERN IM KONTEXT DER SOFTWARE-ENTWICKLUNG

Vorlesungen von: Prof. Dr.-Ing. Anna Sabine Hauptmann

STTI_intro_2017.pdf
Folie 10

5.1 MUSTER VS ANTIMUSTER

Vorlesung
12.06.2017

5.1.1 MUSTER

Wiederverwendung von WISSEN (Strukturen usw.)

- Entwurfs-Muster
  - Erzeuger-Muster
  - Struktur-Muster
  - Verhaltens-Muster
- Architektur-Muster
  - 3-Säulen-Architektur
  - ...

Typische Aufgabe → Lösungsvorschlag durch Muster

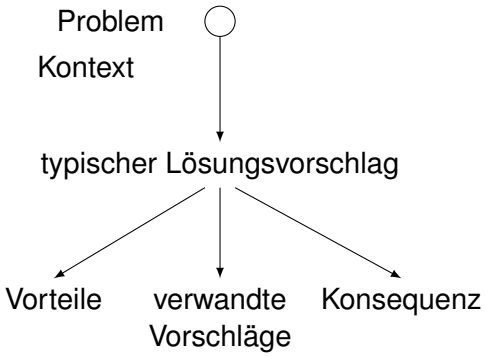
5.1.2 ANTI-MUSTER

Ein Anti-Muster ist eine allgemein verbreitete Lösung einer Aufgabenstellung MIT NEGATIVEN KONSEQUENZEN.
Beschreibung von Anti-Mustern:

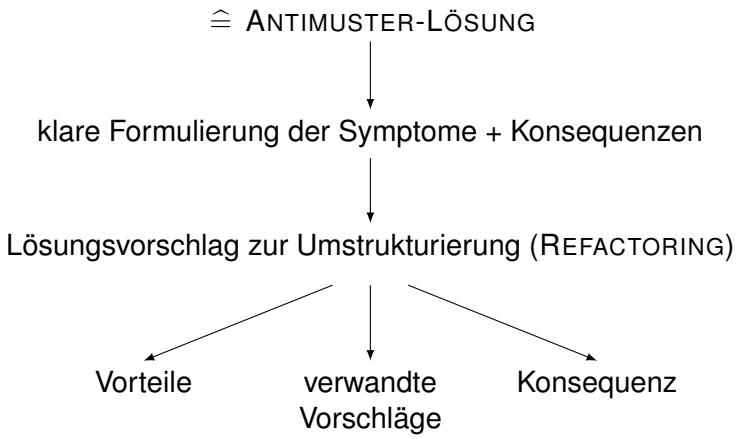
1. typische Aufgabenstellung
2. typische Lösungsvariablen mit negativen Konsequenzen
3. neu strukturieren:
   → REFACTORING

5.1.3 REFERENZ-MODELLE

- Entwurfs-Muster



- Anti-Muster
  Lösung einer Aufgabenstellung mit Symptomen



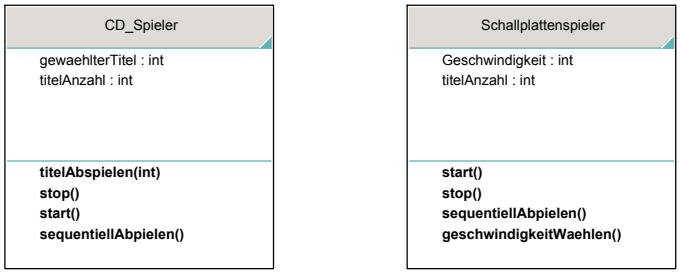
5.2 REFACTORING

→ Umstrukturierung
Möglichkeiten:

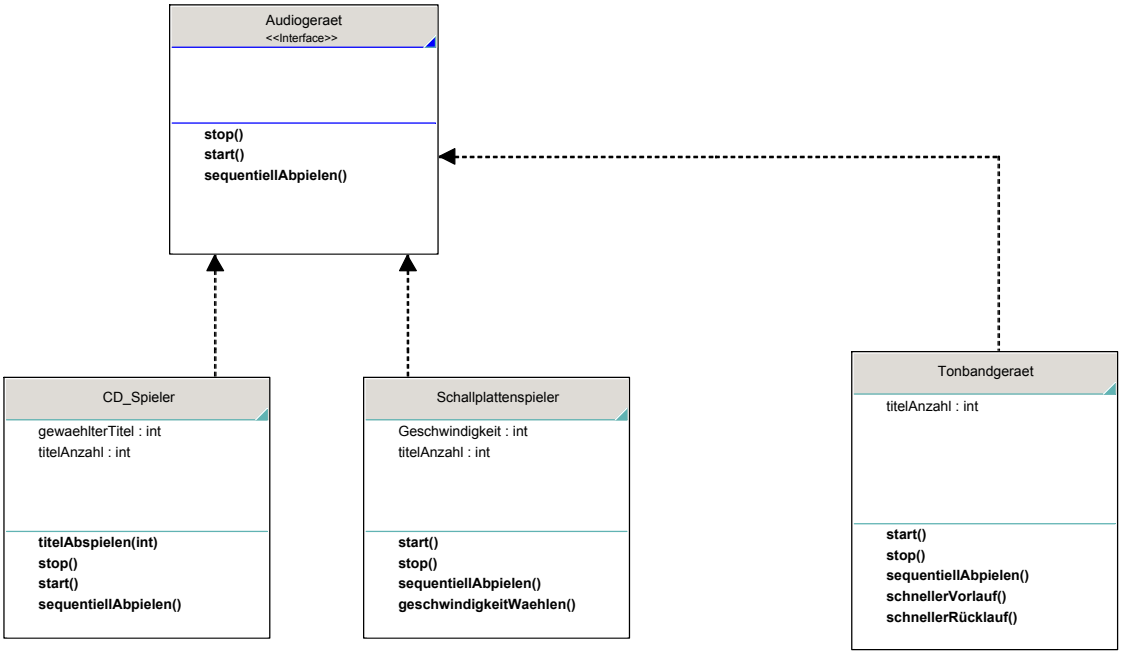
- Funktionalität (Verantwortlichkeit) in andere (neue) Klassen auslagern
- neue Abstraktionsebenen einbauen:
  → Superklasse extrahieren → Generalisierung (Vererbung)
  → Subklassen bilden → Spezialisierung

BEISPIEL

Von:



Zu:



5.3 ANTI-MUSTER

5.3.1 THE BLOB

- Symptom: riesengroße Klasse, die alles allein macht
- Konsequenz: alle Änderungen beziehen sich auf diese Klasse
- Ursache: nicht wirklich objektorientiert programmiert (→ Objekte lösen gemeinsam eine Aufgabe)
- Lösung: Attribute und zugehörige Funktionalitäten/Verantwortlichkeiten heraus finden und in neuer Klasse kapseln

falls Refactoring zu aufwändig:
→ neuer Entwurf (Richtlinien: verteilte Verantwortlichkeit)
z.B.: CRC-Karten (Class Responsibility Collaboraters)
Im Kontrast dazu der „Anti-Blob“: Wenn zu viele Klassen existieren (jede Funktion eine Klasse)

5.3.2 LAVA FLOW

- Symptom: Quellcode der (wahrscheinlich) nicht mehr genutzt wird (→ Dokumentation ???)
- Ursache:
  - Lebensende Entwickler und keine wirkliche Übergabe.
  - prototypische Systeme werden als Produktiv-Systeme weiter entwickelt.
  - Vermischung von Laufzeit- und Persistenzobjekten.
- Lösung: Ist-System dokumentieren → entsprechender Entwurf

5.3.3 POLTERGEISTER

→ Auftauchen und plötzliches Verschwinden

- Symptom: Klasse scheint wichtig, aber keiner weiß genau wofür
- Ursache:
  - redundante Navigationspfade
  - „Helper-“, „Controller-“ Klassen
  - Zustandslose Klassen
  - ...
- Lösung: Poltergeister als Klasse beseitigen. Eventuell den Klassen zuordnen, von denen sie aufgerufen werden.

Vorlesung
22.06.2017

5.3.4 GOLDENER HAMMER

→ „Ich besitze einen Hammer („Wunderwaffe“) und alles andere sind Nägel.“

- Symptome:
  - Die Architektur unseres Systems ist unsere Datenbank.
  - In der Analyse LENKEN die Entwickler die Diskussion zu den Anforderungen aus der Perspektive der technologischen Lösungsmöglichkeiten.
- Ursachen:
  - aktuelle Entwicklung ist nicht im Blick
  - Überschätzung der bisherigen Lösungsvariante
  - blindes Vertrauen
  - hohe Investitionskosten müssen weiterhin die Technologiewahl bestimmen
- Lösung:
  - 2 Aspekte:
    - Vorgehensweise (Kultur, philisophisch)
      → Änderung im Entwicklungsprozess: Freiraum schaffen, Diskussion, Evaluieren
    - SW-System
      Austauschbarkeit durch klare Abgrenzung der Komponenten (Abstraktionsstufe)

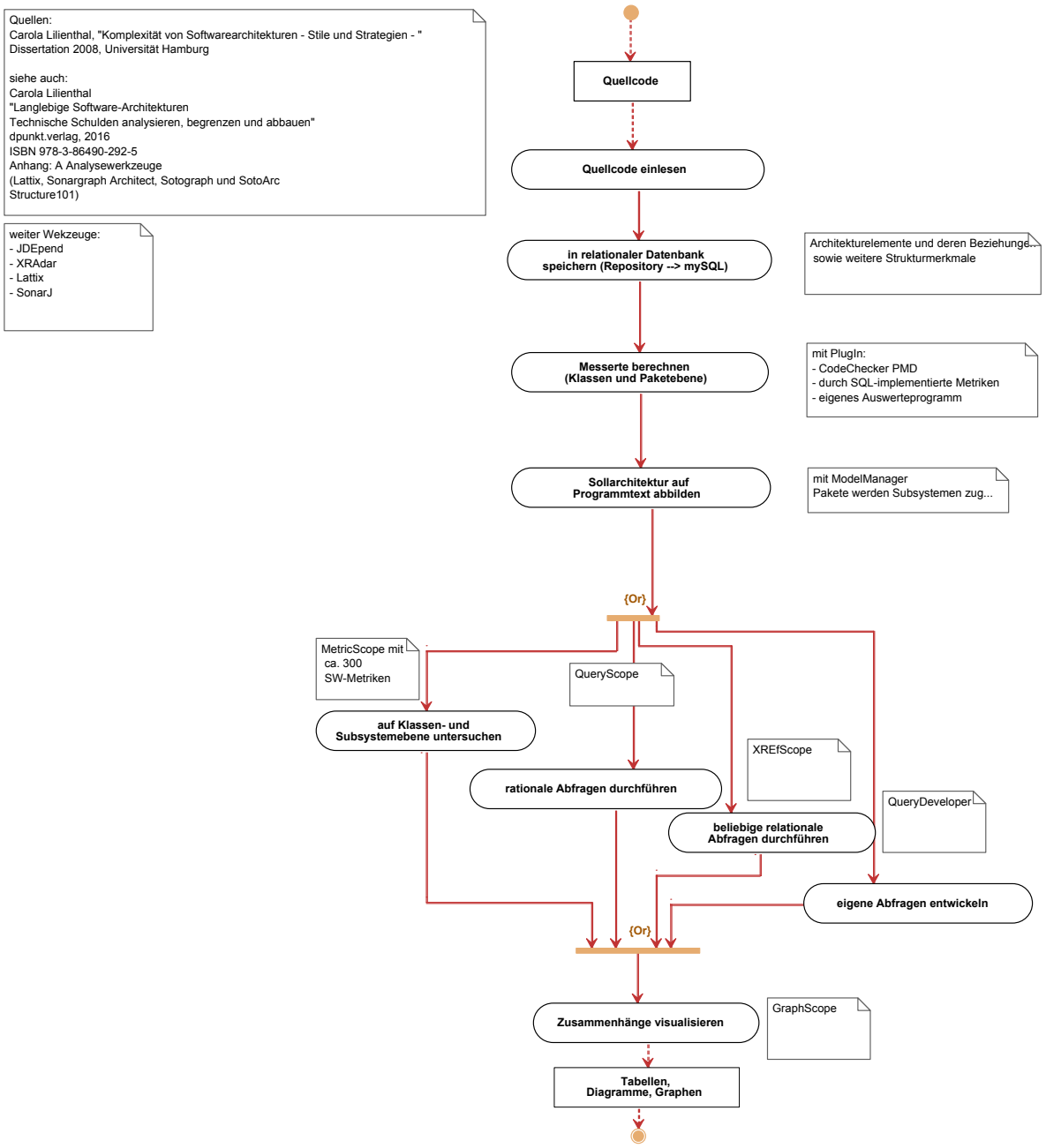
5.3.5 SPAGHETTI-CODE

- Symptome:
  - wenig Struktur im System
  - Objekt sind wie Prozess bezeichnet (mit Verb: bspw. `schreibt` anstatt `Schreiber`)
  - Ablaufsteuerung ist vom Objekt, nicht vom Client des Objektes vorgeschrieben
  - Objektoperationen haben wenig bis keine Parameter (Instanziierung) → statt Parameter:
    - Klassenvariablen
    - globale Variablen

- Ursachen:
  - Sprachkonzepte sind nicht verstanden
  - vor Implementierungsphase gibt es keine Entwurfs-Phase
  - Entwickler arbeiten isoliert voneinander
  - keine Code-Überwachung
- Lösung:
  - Refactoring
    - eher Prävention
    - ständige Code-Überwachung $\rightarrow$ wirkliches Refactoring: z.B. Objekte als Objekte verwenden

5.4 SOTOGRAPH

= Plattform zur schnellen und komfortablen Untersuchung der inneren Struktur von SW-Systemen (für `C / C++, Java, C#, ABAP, PHP, ...`)



5.5 ANTIMUSTER IM KONTEXT DER ARCHITEKTUR

Vorlesung
26.06.2017

Entwurf besteht aus:

1. Architektur fürs GESAMTSYSTEM festlegen ($\rightarrow$ Grobentwurf)
2. Bauplan für EINZELNE KOMPONENTEN erstellen ($\rightarrow$ Feinentwurf)

Die bisherigen Antimuster haben sich mit Fehlern im Feinentwurf beschäftigt.

AUFGABEN DES GROBENTWURFS

3 Dinge, die auf dem Weg zur Architektur erledigt werden müssen:

1. funktionale Gliederung des SW-Systems in grobe Komponenten (Module)
2. Schnittstellen zwischen den Modulen werden definiert ($\rightarrow$ Verbindungsstellen)
3. Technologie zur Implementation der Schnittstellen definieren

ZUR FUNKTIONALEN GLIEDERUNG:

Abstraktionen eines SW-Systems (am Beispiel eines Programms addiert):

- Funktion (addiert und dividiert nicht)
- Daten (addiert Gleitkommazahlen, nicht nur ganze)
- Objekt (führt Funktion und Daten zusammen)
- Zustand

5.5.1 STOVEPIPE, AUTOGENERATED

entsteht ggf. bei der Verteilung von SW-Schnittstellen ($\rightarrow$ verteiltes System): Ein System, das bisher lokal war soll nun zu einem verteilten System werden. Aber was wird verteilt?
Vorsicht vor nicht kommunizierten Annahmen.

5.5.2 STOVEPIPE ENTERPRISE

Wenn eine Software von mehreren genutzt wird, und jeder die Basis braucht, aber jeweils noch ergänzende Dienste benötigt werden:

1. aufgabenspezifische Dienste
2. ergänzende Dienste
3. Basisdienste / Infrastruktur
4. Vorschriften

Dabei sind die Vorschriften und Basisdienste ZENTRAL zu regeln, alles spezifischeres dann DEZENTRAL.
Das sollte sich dann wie eine Pyramide gestalten: Eine breit(ere) zentrale Basis mit schmaler werdenden dezentralen Teilen. Alternativ könnte auf die breite Basis mehrere schmale Teil-Pyramiden mit den spezifischen weiteren Diensten existieren.