Antipatterns

- Dirk Wendling — dw027@hdm-stuttgart.de
- ~~Marc Seeger~~ — ms155@hdm-stuttgart.de
- ~~Stephan Helten~~ — sh094@hdm-stuttgart.de

HOCHSCHULE DER MEDIEN

Antipatterns

Dirk Wendling – DW027
Marc Seeger – MS155
Stephan Helten – SH094

(MIB 4)

Agenda

- Begriffserklärung
- Anti-Patterns im Projektmanagement
- Code-Design-Antipatterns
- Java Antipatterns
- Evolution und AntiPatterns
- how to write unmaintainable code

Begriffserklärung: Antipatterns

- Negativ-Beispiele von bereits durchgeführten Lösungen
- Analyse gibt Hinweise darauf, wie das Problem besser gelöst werden könnte

- Kategorisierung
  - Architektur- bzw. Design-Anti-Patterns
  - Projektmanagement-Anti-Patterns
  - Organisations-, Management- bzw. Prozess-Anti-Patterns
  - Programmierungs-Anti-Patterns
  - Meta-Patterns

Ausblick: Evolution und AntiPatterns

Nacktmull

Kakapo

Welches dieser beiden Tiere ist mittlerweile zum Antipattern geworden?

Die Auflösung

- gibt es am Ende des Vortrags

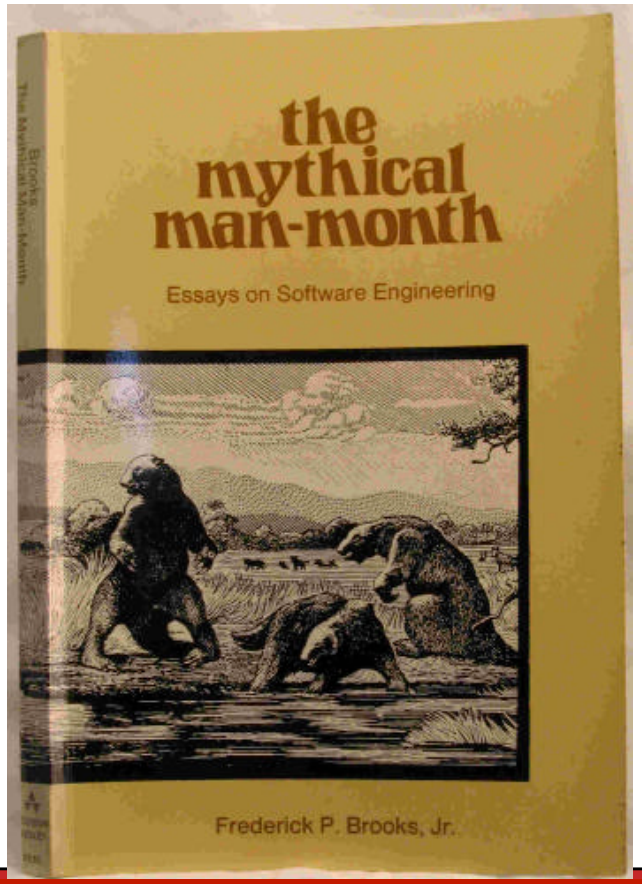
HOCHSCHULE DER MEDIEN

Anti-Pattern im Projektmanagement

Dirk Wendling – DW027
(MIB 4)

Brooksches Gesetz

Fred Brooks, Mythical Man Month:

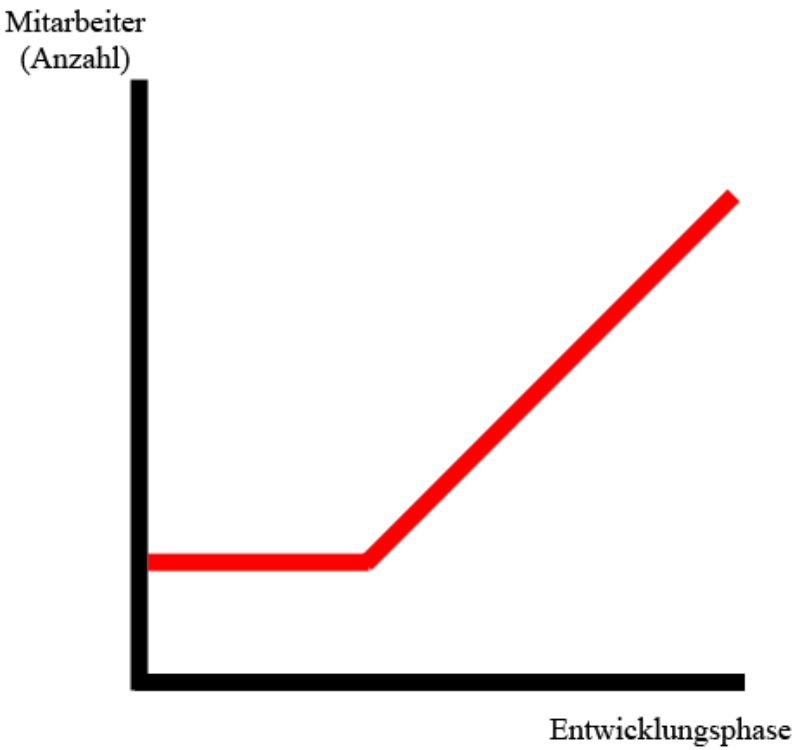


Adding manpower to a late software project makes it later!

Die Einarbeitung des neuen Mitarbeiters kostet wichtige Kapazitäten des produktiv arbeitenden Teams.

→ Projekt verzögert sich weiter

Warme Leiche / Warm body



Frühe Entwicklungsphase:

z.B. System entwerfen

→ wenige Spezialisten

Spätere Entwicklungsphase:

z.B. System umsetzen

→ weitere Spezialisten

„Wenn zwei im Raum einer Meinung sind, ist mindestens einer davon überflüssig.“

Churchill

Single Head Of Knowledge

- Einzelner Mitarbeiter ist alleiniger Wissensträger.
  - Wenn dieser Mitarbeiter weg ist, ist auch das Know-How weg!

Reinventing the wheel

- Software wird immer komplett neu erstellt.
- Es wird keine Rücksicht auf bereits bestehende, ähnliche Software genommen, welche sich bereits bewährt hat.

  → Aufwendig, Teuer, Instabil

Yet Another Meeting Will Solve It

- Ein Meeting in einem verspäteten Projekt einzuberufen, bedeutet nur, dass sich dieses noch weiter verzögert.

- → „Noch ein Meeting mehr wird es lösen.“
  - NICHT!

Ecosystem	Legacy	Scope	Service Oriented	Market Window
Think Outside the Box	Network[ing]	Diversity	Multitask[ing]	Action Item
Dynamic	Target Audience	BULLSHIT BINGO (free square)	Off Site Meeting	Out Source
Total Quality [or] Quality Driven	Issues	Prototype	No-Brainer	Gap Analysis
Followup	Off-line	Pushback	Exposure	Pipeline

Negative Producing Programmer

- Einen unproduktiven Entwickler aus einem Team zu entfernen kann die Produktivität mehr erhöhen,
  als einen guten Entwickler hinzuzufügen.

Corncob / Quertreiber (1/2)

- Ein Quertreiber denkt an erster Stelle an sich.
- Er strebt nach Anerkennung und finanziellen Vorteilen.

- Motive des Quertreibers:
  - Habgier
  - Stolz
  - Engstirnigkeit

- Symptome:
  - Mangelnder Fortschritt im Projekt

Corncob / Quertreiber (2/2)

- Gründe für das Entstehen von Quertreibern:
  - Management geht nicht auf das Verhalten des Querkopfes ein
  - Quertreiber verfolgt eigene Ziele ←oft ungleich→ Projektziel

Refactoring:

- Ihn zur Verantwortung ziehen:
  - „You raissed the issue, you own/fix the problem“
- Klärendes Gespräch mit dem Management
- Pizza Party
  - „There is no problem so serious that a pizza party can‘t resolve“ (Randall Oakes)

Feature creep

- Erschleichung von immer weiteren Funktionalitäten durch den Kunden.

  → Budget wird überschritten

  →Projekt verzögert sich immer weiter

- Abhilfe:
  - Genau definierte Anforderungsliste bzw. Pflichtenheft.

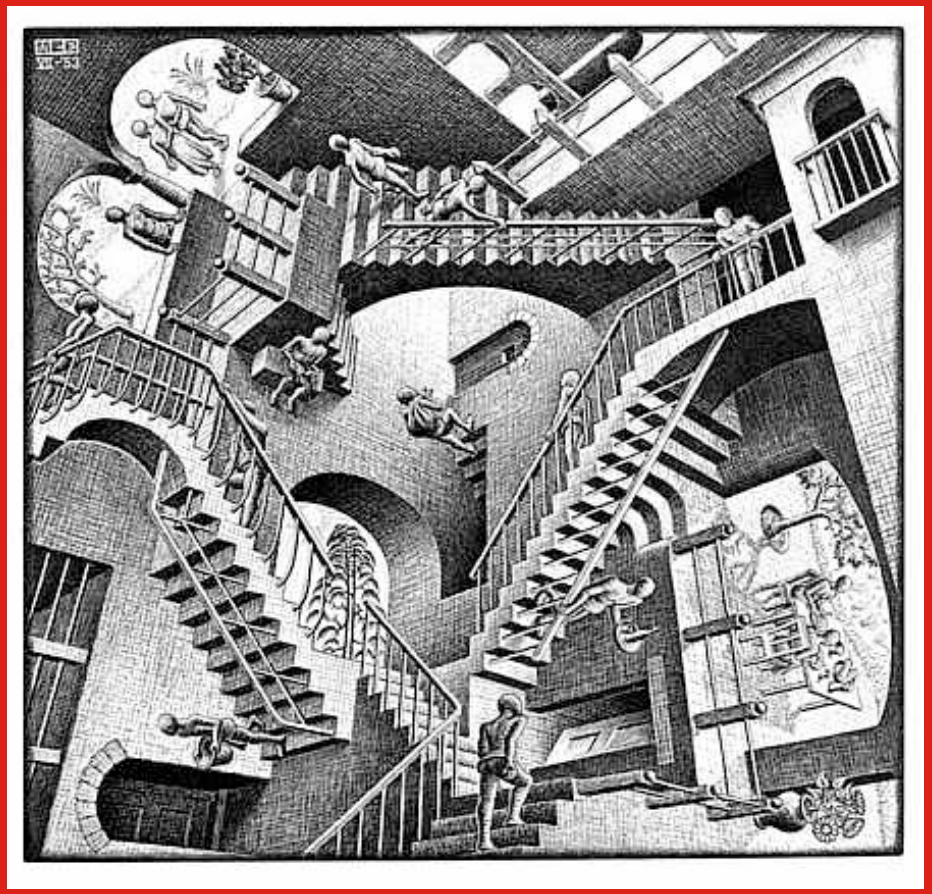
Death Sprint

- „Anstrengung bis zum letzten Atemzug“
- Bei der Einhaltung eines überhitzten Zeitplanes beginnen die Entwickler zu schlampen.
- Äußerlich macht sich dies zu Beginn kaum bemerkbar doch die Qualität der Software leidet maßgeblich.
  - Es wird nur noch versucht die Funktionen fertig zu stellen, ohne nach einer schlankeren/besseren Lösung derer zu arbeiten.
  - Keine Dokumentation od. Qualitätssicherung
    → führt früher oder später zu Unwartbarkeit
- Beispiel: Browser War

Golden Hammer

- Ein bevorzugter Lösungsweg wird fälschlicherweise als universell anwendbar angesehen.
- Problem:
  - Der Lösungsweg passt nur scheinbar zur Aufgabenstellung.
- „Wenn mein einziges Werkzeug ein Hammer ist, sieht jedes Problem wie ein Nagel aus.“ (Abraham Maslow)

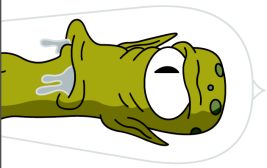
Design/Architektur-Antipatterns



Marc Seeger – MS155
(MIB 4)

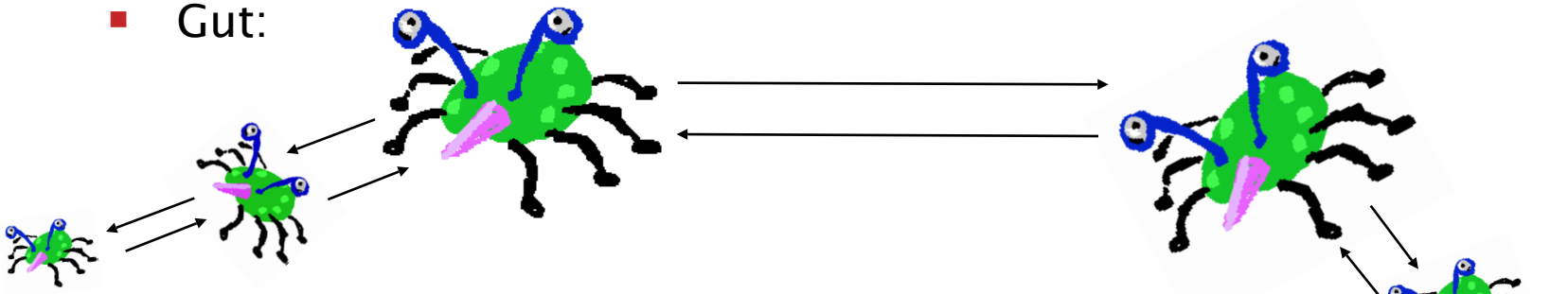
Alien Spiders!

- Problem: Kommunikation läuft nicht über definierte Nachrichtenkanäle
- Unnötige Vernetzungen zwischen Modulen
- „Law of Demeter“
- Jedes Modul sollte stets nur sehr begrenztes Wissen über andere Module haben, also nur mit seinen direkten Nachbarn kommunizieren.
- Lösung: Fernverbindungen müssen beim Refactoring entfernt werden

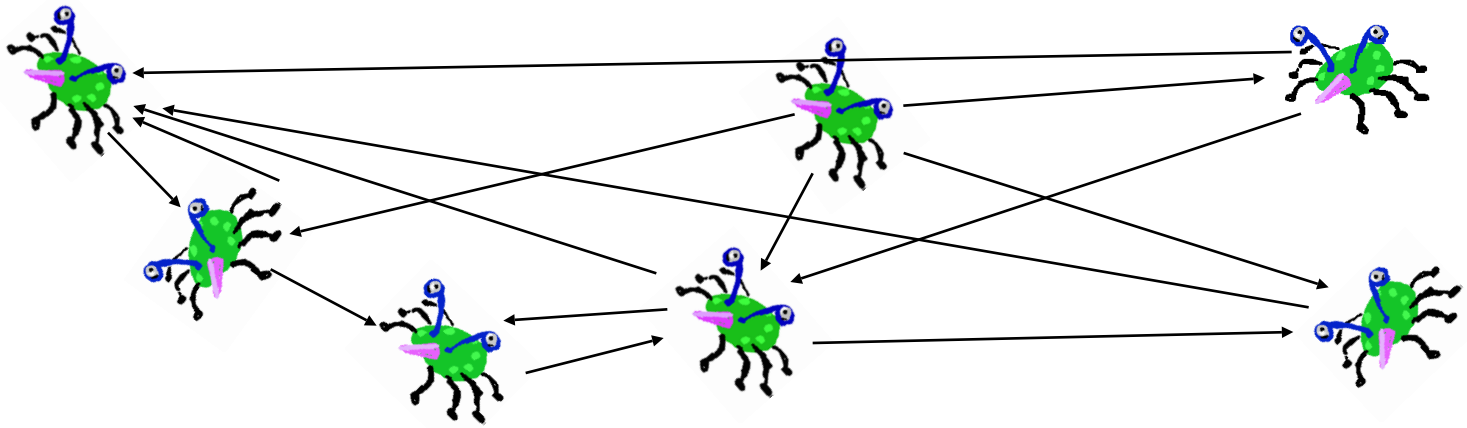


Alien Spiders!

- Gut:



- Schlecht:

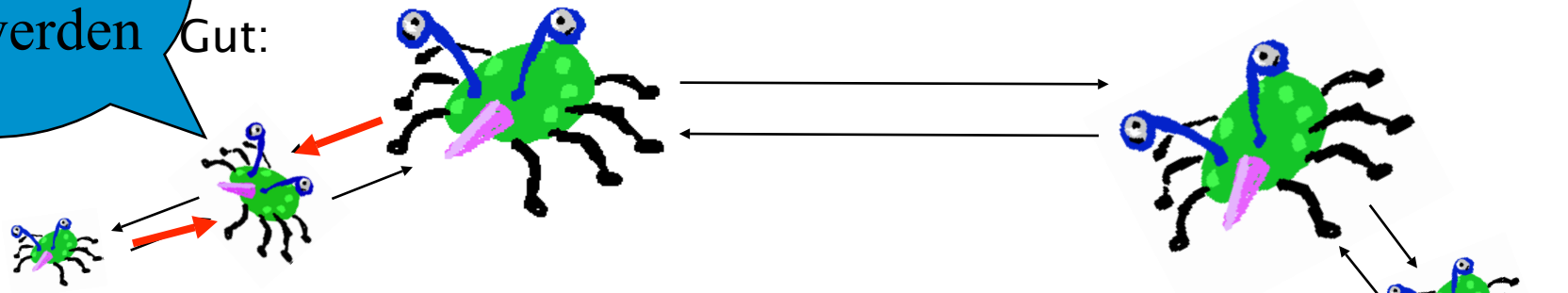


lien Spiders!

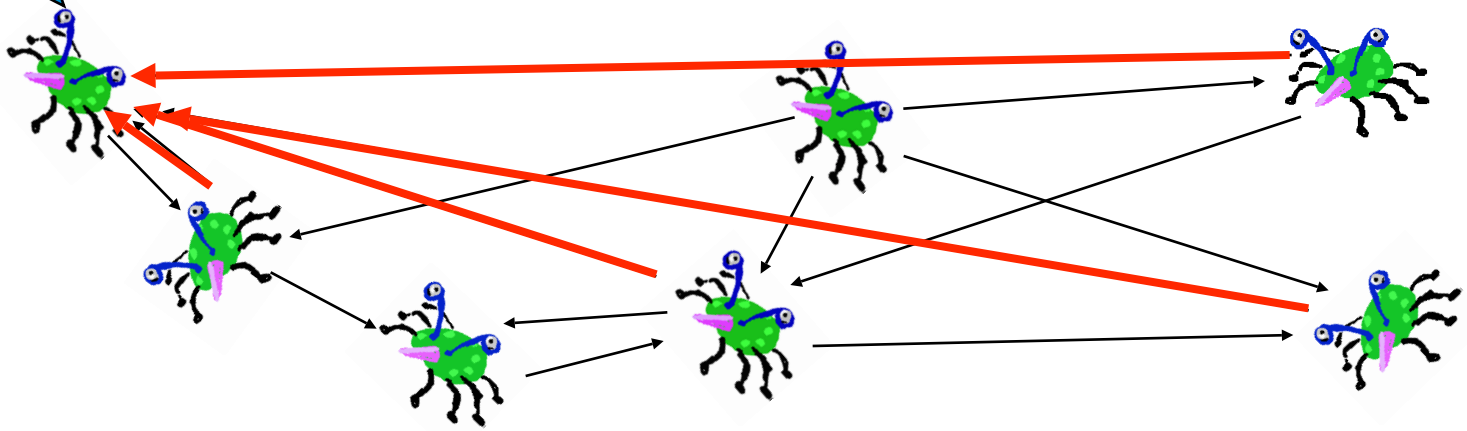
Ich muss ausgewech selt werden

Ich auc h

- Gut:



- Schlecht:

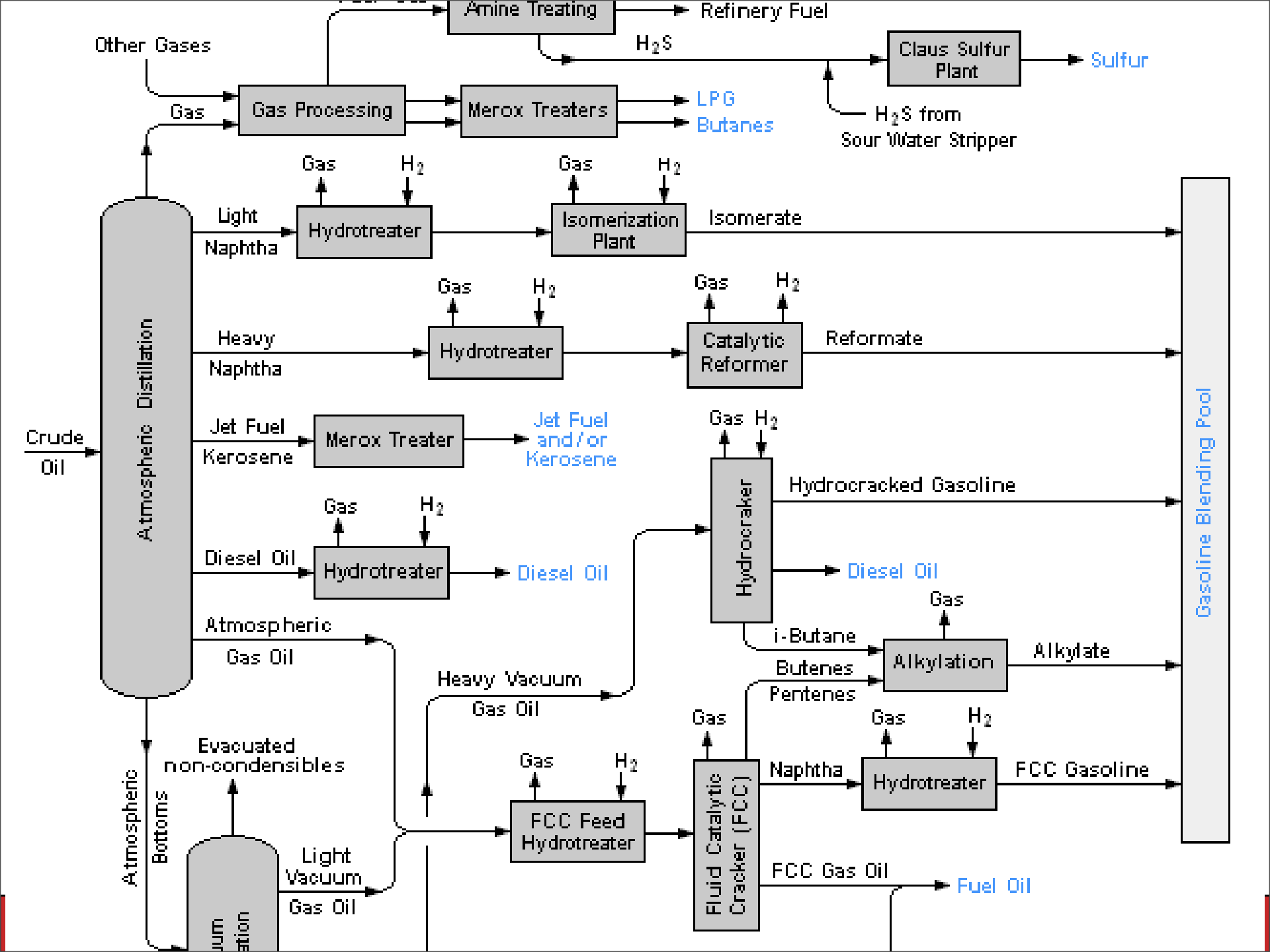


Gas Factory

„Mit Kanonen auf Spatzen schießen“

Gas Factory

- Unnötig Komplexes Design
- Grund:
  - mangelnde Abstraktion
  - mangelnder Überblick über Gesamtproblem


Amine Treating
Refinery Fuel
Other Gases
H_2S
Claus Sulfur Plant
Sulfur
Gas
Gas Processing
Merox Treaters
LPG
Butanes
H_2S from Sour Water Stripper
Gas
H_2
Light Naphtha
Hydrotreater
Isomerization Plant
Isomerate
Heavy Naphtha
Catalytic Reformer
Reformate
Crude Oil
Atmospheric Distillation
Jet Fuel Kerosene
Merox Treater
Jet Fuel and/or Kerosene
Diesel Oil
Hydrocracker
Hydrocracked Gasoline
Atmospheric Gas Oil
i-Butane
Alkylation
Alkylate
Butenes
Pentenes
Heavy Vacuum Gas Oil
Evacuated non-condensibles
Atmospheric Bottoms
FCC Feed Hydrotreater
Fluid Catalytic Cracker (FCC)
Naphtha
FCC Gasoline
Light Vacuum Gas Oil
FCC Gas Oil
Fuel Oil
Gasoline Blending Pool

Gas Factory – Hallo Welt!

Lisp:

```
(alert "Hallo Welt!")
```

Windows C API:

```
#include <windows.h>

   LRESULT CALLBACK WindowProcedure
(HWND, UINT, WPARAM, LPARAM);

   char szClassName[] = "MainWnd";
   HINSTANCE hInstance;

   int WINAPI WinMain(HINSTANCE hInst,
HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow)
   {
     HWND hwnd;
     MSG msg;
     WNDCLASSEX wincl;

     hInstance = hInst;
```

Big Ball of mud

- "A Big Ball of Mud is a haphazardly structured, sprawling, sloppy, duct-tape-and-baling-wire, spaghetti-code jungle“

Poltergeist

- „pop in to make something happen“

Poltergeist

- Ein Poltergeist hat
  - geringe Verantwortung
  - kurze Lebensdauer
  - keinen Zustand
  - keine Aufgabe außer Aktionen in anderen Klassen zu initialisieren

Poltergeist

- Folgen:
  - Wartung sehr schwer möglich
  - Bläht mein Design unnötig auf
  - Unnötige Redundanz zwischen Klassen
  - Vernichtet Ressourcen

Poltergeist

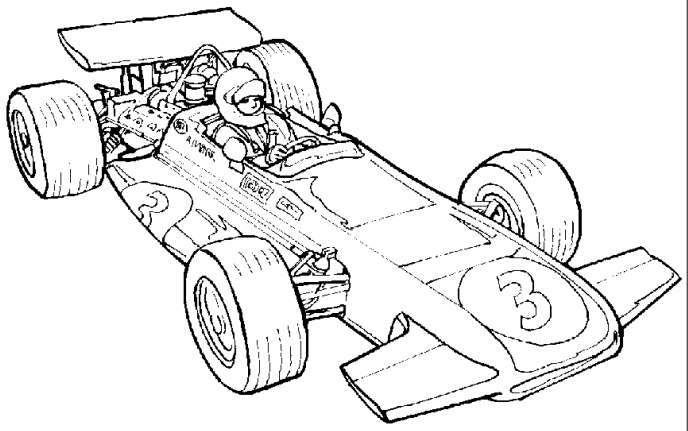
- Gründe:
  - Entwickler ohne OO Kenntnisse
  - OO ergibt keinen Sinn für die Aufgabe
    - „There is no right way to do the wrong thing.“
  - Architekturfestlegung in Requirement-Analyse

Poltergeist

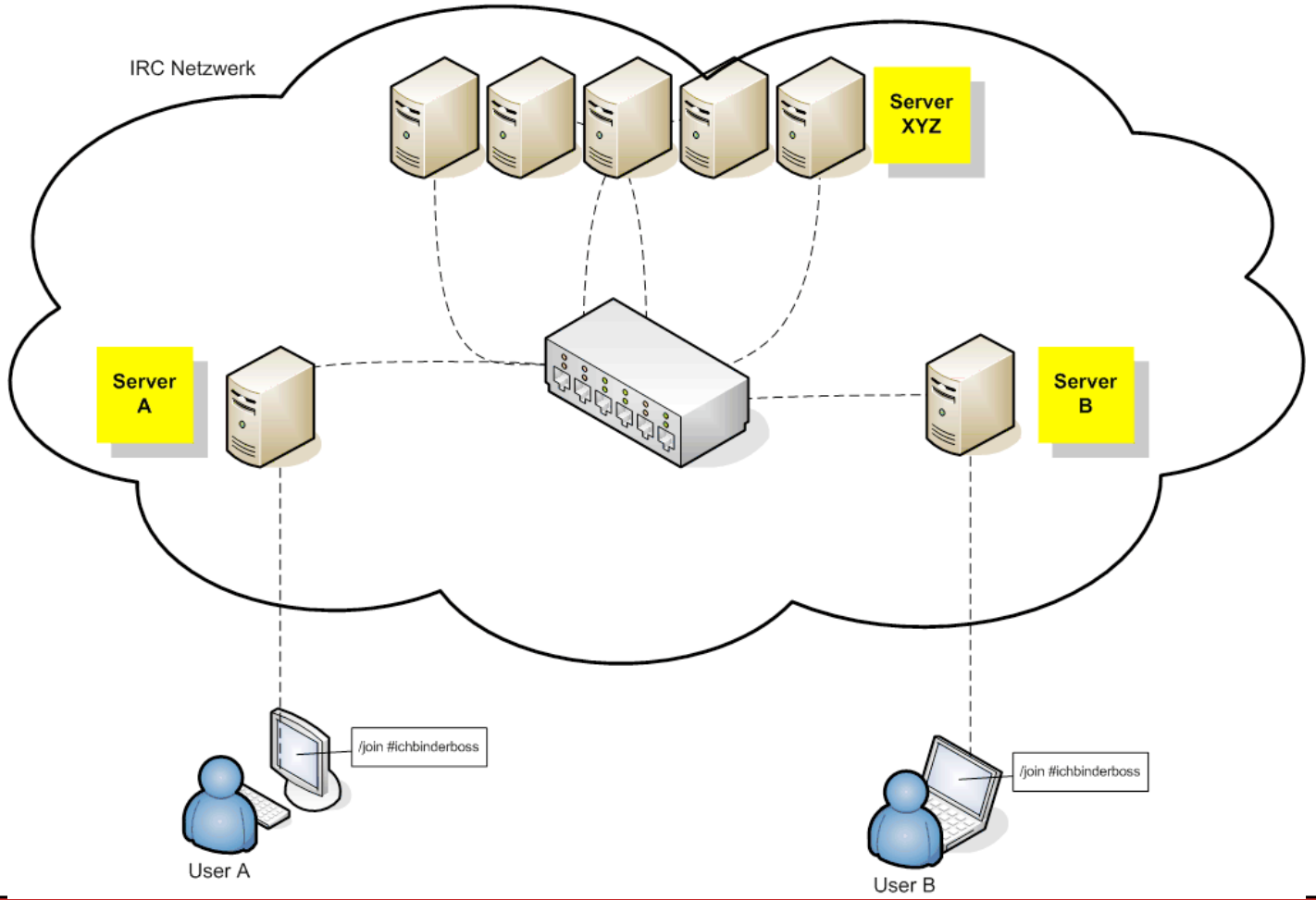
- Lösungsansatz:
  - Polgergeistklasse entfernen
  - Funktionalität in beteiligte Klasse verschieben

Race Condition

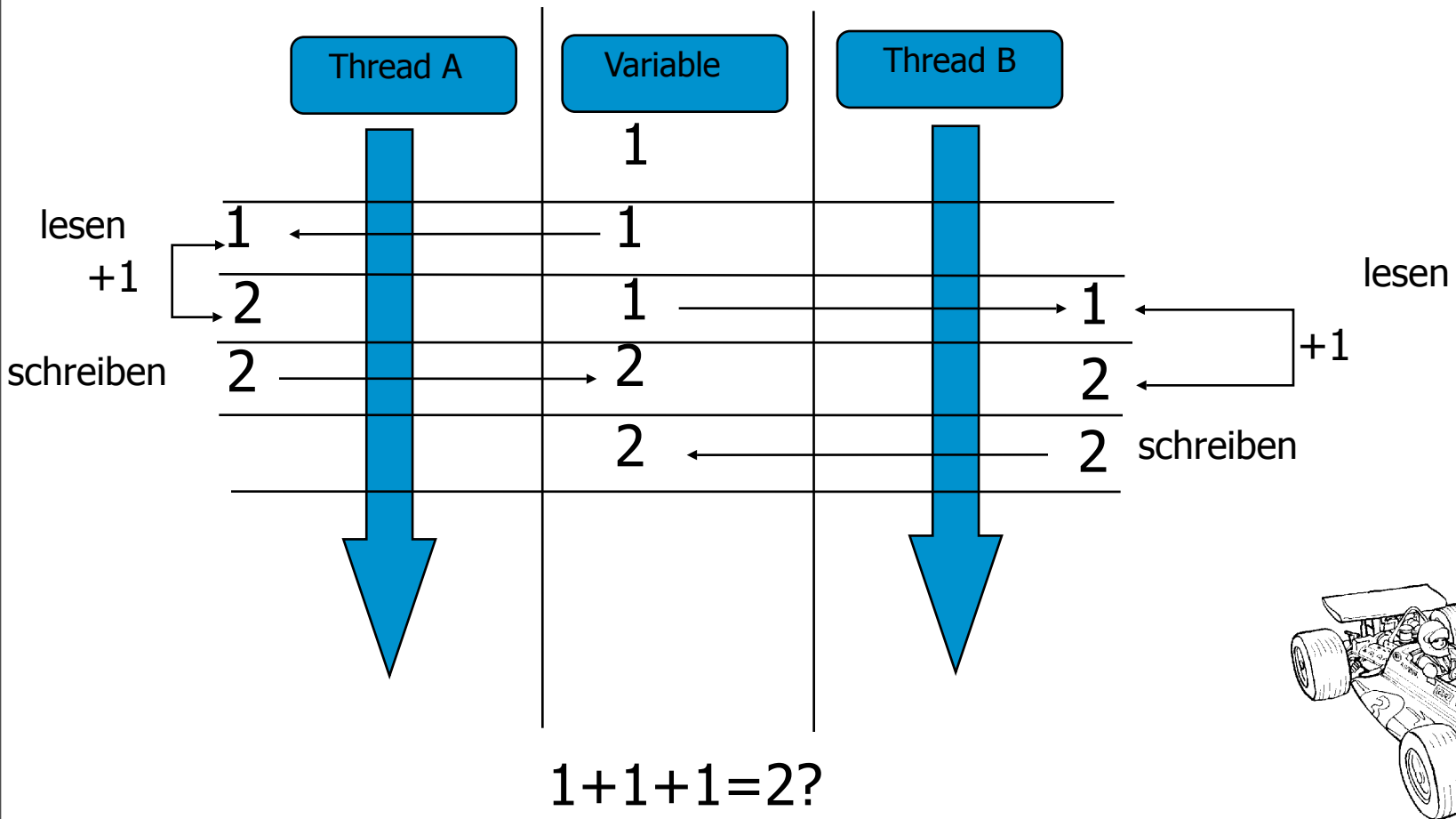
- Problem:
  - Ergebnis einer Operation ist abhängig vom zeitlichen Ablauf ihrer Einzelschritte

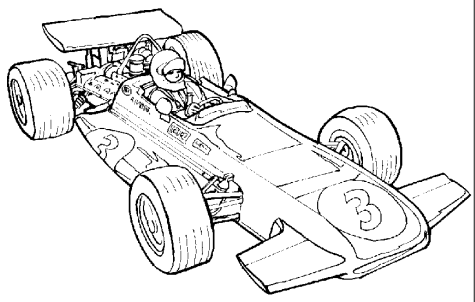


Race Condition



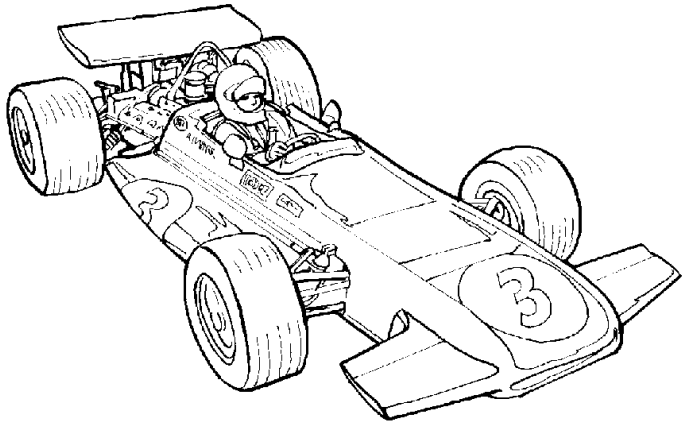
Race Condition





Race Condition

- Lösung:
  - Vorlesung 21801: Verteilte Systeme
  - Ring-Puffer, Spinlocks, …



Design by Commitee

- „ a camel is a horse designed by committee “

Design by Commitee

- Grundlage für „Design by Commitee“ in der IT Branche:
  - Meeting von Leuten die keine Ahnung haben von:
    - Code
    - Implementierung
    - Design
  - Aber:
    - Einige (veraltete) Entwicklungsmodelle kennen

Design by Commitee

- Ergebnis:
  - Ein Design welches folgende Punkte hat:
    - Unnötige Komplexität
    - interne Unstimmigkeiten
    - Logische Fehler
    - Banale Inhalte
    - Keine gemeinsame Vision
    -

Design by Commitee

Design by Commitee

Minivan

HOCHSCHULE DER MEDIEN

Java-Antipatterns

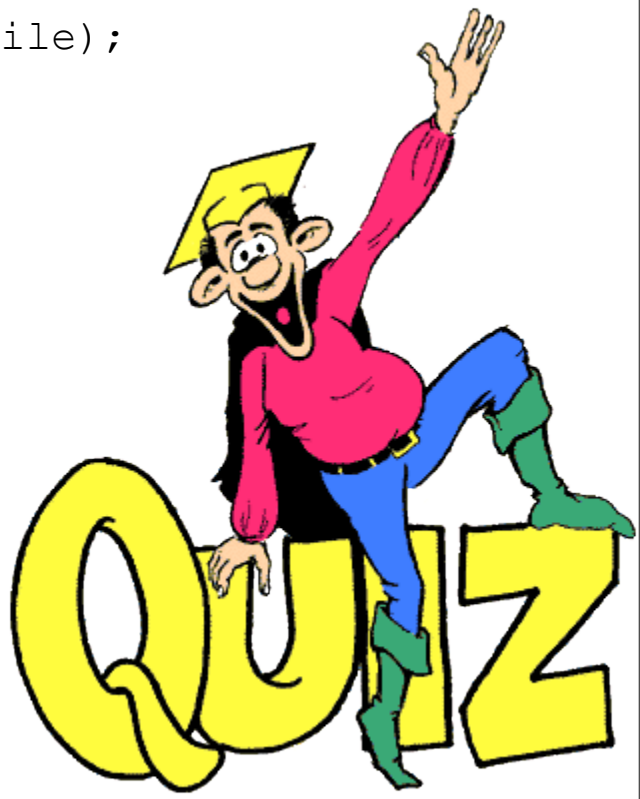
Stephan Helten - SH094
(MIB 4)

Quiz: Frage 1

So bitte nicht:

```
InputStream in = new FileInputStream(file);
int b;
while ((b = in.read()) != -1) {
    ...
}
```

Aber warum?



unbuffered streams

```
InputStream in = new FileInputStream(file);
int b;
while ((b = in.read()) != -1) {
    ...
}
```

- Code liest File byteweise ein
- Jeder read()-Call verursacht Zugriff aufs Filesystem über das Java Native Interfae
- JNI-Calls sind TEUER!
- Dauer für 1 MB-File von Festplatte lesen: 1 Sekunde

buffered streams

- merke: BUFFER ist BESSER!

```
InputStream in = new BufferedInputStream(new
   FileInputStream(file));
```

- BufferedInputStream() reduziert JNI-calls
- Dauer für 1 MB-File von Festplatte lesen: 60 Mikrosekunden
  - Ersparnis von 94% zu vorher
- auch für OutputStreams und Sockets, etc. …

Quiz: Frage 2

So bitte nicht:

```
public class B {
    private int count = 0;
    private String name = null;
    private boolean important = false;
}
```

Aber warum?

overkill initialization

```
public class B {
    private int count = 0;
    private String name = null;
    private boolean important = false;
}
```

- Class initializer wird vorm Konstruktor aufgerufen um Werte zu setzen:
  - unnötiger overhead

faster initialization

```
public class B {
    private int count;
    private String name;
    private boolean important;
}
```

- Class initializer wird nicht aufgerufen
- möglich, da Java Initialisierungswerte immer gleich setzt, wenn sie nicht angegeben sind:
  - int: 0
  - String: null
  - boolean: false

Quiz: Frage 3

So bitte nicht:

```
Query q = ...
Person p;
try {
    p = (Person) q.getSingleResult();
} catch(Exception e) {
    p = null;
}
```

Aber warum?

catch all: I don't know the right runtime exception

```
Query q = ...
Person p;
try {
    p = (Person) q.getSingleResult();
} catch(Exception e) {
    p = null;
}
```

- **Drei Exceptions können auftreten:**
  1. **Ergebnis nicht eindeutig**
  2. **Kein Ergebnis vorhanden**
  3. **Fehler beim Absetzen des queries**
- **Alle werden gleich behandelt (p=null)**

only catch the right runtime exception

```
Query q = ...
Person p;
try {
    p = (Person) q.getSingleResult();
} catch(NoResultException e) {
    p = null;
}
```

- **Nur im Fall, dass kein Ergebnis gefunden wurde macht die Fehlerbehandlung (p=null) Sinn**
- **Bei mehreren Ergebnissen oder dem fehlerhaften query wird p nicht gleich null gesetzt**

Quiz: Frage 4

So bitte nicht:

```
try {
   ... do risky stuff ...
} catch(SomeException e) {
   // never happens
}
... do some more ...
```

Aber warum?

the exception that never happens

```
try {
   ... do risky stuff ...
} catch(SomeException e) {
   // never happens
}
... do some more ...
```

- Programmierer glaubt, dass in seiner speziellen Situation diese Exception nicht auftreten kann.
- Falls sie irgendwann doch auftritt, merkt er das nicht

the exception that never happens hopefully

```
try {
   ... do risky stuff ...
} catch(SomeException e) {
   // never happens hopefully
   throw new IllegalStateException(e.getMessage(),
    e); // crash early, passing all information
}
... do some more ...
```

- **Wenn diese Exception irgendwann doch einmal auftritt, dann wird die Fehlerursache detailliert ausgegeben**
- **Wartbarkeit und Fehlersuche erleichtert**
- **Falls Sie wirklich nie auftritt: Kein Nachteil**

Quiz: Frage 5

```
Class beanClass = ...
//Ist TestBean eine SuperKlasse?
if (beanClass.newInstance() instanceof TestBean)
```

Warum so nicht?

poor use of reflection

```
Class beanClass = ...
//Ist TestBean eine SuperKlasse?
if (beanClass.newInstance() instanceof TestBean)
```

- **Eine Instanz von einer unbekannten Klasse zu erstellen ist gefährlich, denn:**
  - **Man weiss nicht was der Konstruktor tut**
  - **Erstellen der Instanz kann teuer sein**
  - **Der Default-Konstruktor könnte fehlen**
    - Exception tritt auf

smart use of reflection

```
Class beanClass = ...
if (TestBean.class.isAssignableFrom
   (beanClass)) ...
```

- **Instanz wird nicht erzeugt**
- **Semantik mit instanceof identisch**

Quiz: Frage 6

```
Collection l = new Vector();
for (...) {
    l.add(object);
}
```

Warum ist das ein Antipattern?

synchronization overkill

```
Collection l = new Vector();
for (...) {
    l.add(object);
}
```

- Vector = synchronized ArrayList
- Hashtable = synchronized HashMap
- Wenn sowieso nur ein Thread zugreift, dann ist Vector zu verwenden ein SYNCHRONIZATION OVERKILL

avoiding synchronization overkill

```
Collection l = new ArrayList();
for (...) {
    l.add(object);
}
```

- **ArrayList hat gleichen Funktionsumfang wie Vector**
- **Zugriff muss nicht synchronized sein, da single-threaded**
- **Deutlicher Performance-Vorteil gegenüber der ersten Lösung**

Quiz: Frage 7

```
//Auszug aus einer Entity Bean
private String name;

public void setName(String name) {
    this.name = name.trim();
}

public void String getName() {
    return this.name;
}
```

Warum ist der Einsatz von trim() hier ein Antipattern?

modifying setters

```
public void setName(String name) {
    this.name = name.trim();
}
```

- Es werden andere Daten gespeichert als dem Setter übergeben werden
  - Getter liefert andere Daten als Setter setzt
- Entity Beans sind nicht für die Geschäftslogik, sondern nur für die Datenhaltung zuständig
- Kann unvorhersehbare Nebeneffekte hervorrufen, dessen Ursache nur schwer zu finden ist

modify before set

```
person.setName(textInput.getText().trim());
```

- **Geschäftslogik von Datenhaltung getrennt**
- **Keine unerwünschten Nebeneffekte**
- **Transparenz auch für andere Programmierer**

Quiz: Letzte Frage

```
//Methode soll für Files bis 2 GB die Größe zurückgeben
public int getFileSize(File f) {
  long l = f.length();
  return (int) l;
}
```

Sieht doch gut aus – Ist es das wirklich?

not noticing overflows

```
public int getFileSize(File f) {
  long l = f.length();
  return (int) l;
}
```

- Bei Files, die größer als 2 GB sind, wird durch das Casten ein falscher Wert zurückgeliefert

check for a possible overflow

```
public int getFileSize(File f) {
  long l = f.length();
  if (l > Integer.MAX_VALUE)
     throw new IllegalStateException("int overflow");
  return (int) l;
}
```

- Methode überprüft auf möglichen Overflow und wirft bei Bedarf eine Exception
- Keine falschen Ergebnisse!

HOCHSCHULE DER MEDIEN

Evolution und Antipatterns

Stephan Helten - SH094
(MIB 4)

Evolution und AntiPatterns

Nacktmull

Kakapo

Welches dieser beiden Tiere ist mittlerweile zum Antipattern geworden?

Evolution und AntiPatterns

Nacktmull

Kakapo

Welches dieser beiden Tiere ist mittlerweile zum Antipattern geworden?

Der Kakapo

- Ist ein Papagei, der vom Aussterben bedroht ist (es gibt 85 Stück)
- Besiedelt die drei neuseeländischen Hauptinseln
- Seine Anpassungen an Neuseelands Inseln wurden ihm zum Verhängnis, als der Mensch kam

Der Kakapo und das Fliegen

Der Kakapo…

- hat das Fliegen verlernt, er kann nurnoch gleiten
- kann aber gut laufen
- kugelrund und gut genährt
- kann gut klettern

Der Kakapo und seine Feinde

Der Kakapo…

- ist sehr neugierig
- Hat kein Feindverhalten gegen Bodenprädatoren
- erstarrt bei Gefahr und verlässt sich auf seine Tarnung
  - das ist gut gegen Adler
  - das ist schlecht gegen Raubtiere und Menschen

Die Kakapo-Männchen

Die Kakapo-Männchen…

- rufen bis zu 8 Stunden pro Nacht drei bis vier Monate lang Ihre Balzrufe und verlieren in dieser Zeit ca. die Hälfte ihres Körpergewichts
- fangen erst ab dem 5. Lebensjahr mit Balzrufen an
- paaren sich auch gerne mit Ästen und zusammengerollten Pullovern

Die Kakapo-Weibchen

Die Kakapo-Weibchen…

- lassen jede Nacht für die Nahrungssuche ihre Eier alleine zurück
  - die Eier unterkühlen
  - die Eier werden leichte Beute für andere Tiere
- Weibchen brüten nur, wenn der Rimu-Baum besonders intensiv blüht
  - das geschieht nur alle drei bis fünf Jahre

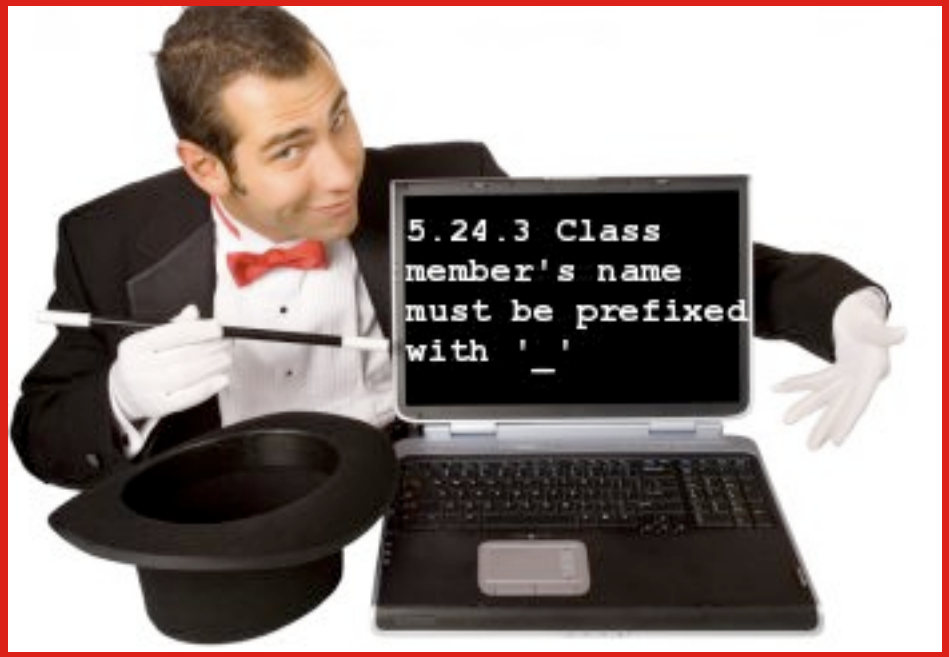
- Weibchen sind erst ab dem 10. Lebensjahr zur Paarung bereit

Fazit

- Der Kakapo ist gut an seine Umwelt angepasst gewesen, bevor der Mensch und mit ihm seine Haustiere und andere Prädatoren auf die Inseln kamen
- Durch die neuen Umweltbedingungen hat sich sein, durch die Evolution entstandenes, Verhaltensmuster zum Antipattern umgewandelt

- auf die Software-Entwicklung übertragen:
  - Design-Patterns können sich durch verschiedene Umstände zu Antipatterns wandeln
  - Design-Patterns sind nicht für alle Zeiten gültig:
    - Was gestern noch üblich und gut war kann morgen schon ein Antipattern sein

How to write unmaintainable code



Marc Seeger – MS155
(MIB 4)

Namensgebung

"When I use a word," Humpty Dumpty said, in a rather scornful tone, "it means just what I choose it to mean - neither more nor less."

Lewis Carroll - Through the Looking Glass, Chapter 6

Namensgebung

- Kreative Schreibfehler!
  - getNumber <--> setNubmer()
- Abstraktionsvermögen!
  - PerformDataFunction(), DoSomething(), HandleStuff()
- GrossBuchStaben
- ASCII ist euer Freund
  - Ð, ñ, ¥, µ
- Emotionen!
  - marypoppins = (superman + starship) / god;

Tarnung

The longer it takes for a bug to surface, the harder it is to find.
- Roedy Green

Tarnung

```
for(j=0; j<array_len; j+ =8)
    {
    total += array[j+0 ];
    total += array[j+1 ];
    total += array[j+2 ]; /* Main body of
    total += array[j+3]; * loop is unrolled
    total += array[j+4]; * for greater speed.
    total += array[j+5]; */
    total += array[j+6 ];
    total += array[j+7 ];
    }
```

```
array = new int []
    {
    111,
    120,
    013,
    121,
    };
```

Dokumentation

Incorrect documentation is often worse than no documentation.
- Bertrand Meyer

Dokumentation

- Lügt in euren Kommentaren (Man muss ja nicht immer „up-to-date“ sein)
- Dokumentiert nur das „wie“, nicht das „warum“
- „makeSnafucated()“ --> /* make snafucated */

Programm Design

The cardinal rule of writing unmaintainable code is to specify each fact in as many places as possible and in as many ways as possible.

- Roedy Green

Programm Design

- Java Casts
  - Datentyp ändern --> alle Casts müssen geändert werden
- Riesen Listener
  - Warum für jeden Button einen Listener wenn die sich auch alle einen teilen können :)
- Public deklarieren und setter-Methoden
- Es gibt nur ein wahres Layout - kein Layout (absolute Werte)
- „Too Much Of A Good Thing“

```
myPanel.add( getMyButton() );
private JButton getMyButton()
    {
    return myButton;
    }
```

Programmiersprachen

nutzt LISP:

```
(lambda (*<8-]= *<8-[= ) (or *<8-]= *<8-[= ))
(defun :-] (<) (= < 2))
(defun !(!)(if(and(funcall(lambda(!)(if(and '(< 0)(< ! 2))1 nil))(1+ !))
(not(null '(lambda(!)(if(< 1 !)t nil)))))1(* !(!(1- !)))))
```

Unsere Quellen

- Projektmanagement & Software-Design-Antipatterns:
  http://www.little-idiot.de/teambuilding/AntiPatternSoftwareentwicklung.pdf
- Java-Antipatterns:
  http://www.odi.ch/prog/design/newbies.php
- how to write unmaintainable code:
  http://thc.org/root/phun/unmaintain.html
- Kakapo:
  Douglas Adams – Die Letzten ihrer Art
  [Heyne (November 1992) | ISBN-10: 3453061152]

Vielen Dank für Eure Aufmerksamkeit!

HOCHSCHULE DER MEDIEN

Hochschule der Medien

Nobelstraße 10
70569 Stuttgart

Tel. 0711 8923 10
Fax 0711 8932 11

info@hdm-stuttgart.de