

IT-Kennzahlen und Softwaremetriken

- Proseminar -

im
Sommersemester 2010

- Die Zyklomatische Komplexität -

Ein Komplexitätsmaß

Gliederung Teil 1

- 1) Wichtigkeit eines Komplexitätsmaßes
- 2) Die Zyklomatische Komplexität
 - 1) Grundlagen
 - 2) Die Zyklomatische Zahl
 - 3) Ergebnisse
- 3) Anwendung und Erfahrung in der Realität
- 4) Vereinfachungen
 - 1) Anhand der Formel
 - 2) Anhand des Kontrollflussgraphen

Gliederung Teil 2

- 5) Strukturlose Programmierung
- 6) Testmethodik

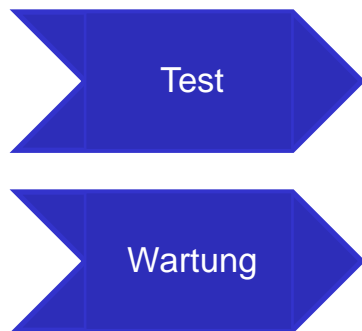
Gliederung Teil 1

- 1) Wichtigkeit eines Komplexitätsmaßes**
- 2) Die Zyklomatische Komplexität
 - 1) Grundlagen
 - 2) Die Zyklomatische Zahl
 - 3) Ergebnisse
- 3) Anwendung und Erfahrung in der Realität
- 4) Vereinfachungen
 - 1) Anhand der Formel
 - 2) Anhand des Kontrollflussgraphen

1) Wichtigkeit eines Komplexitätsmaß



Stark vereinfachter Ablauf der Softwareentwicklung



**50 % des
Entwicklungsaufwands**

**10 - 30 % der gesamten
Softwareinvestitionskosten**

1) Wichtigkeit eines Komplexitätsmaß

- Weniger komplexe Programme würden also Zeit und Geld sparen
- Eine **quantitative Analyse** wäre von Vorteil

Klar ist auch:

- Die Komplexität z.B. beim Testen eines Programms oder Moduls hängt von der **Struktur** ab
- Die Physikalische Größe spielt keine Rolle

Gliederung Teil 1

- 1) Wichtigkeit eines Komplexitätsmaßes
- 2) Die Zyklomatische Komplexität**
 - 1) Grundlagen**
 - 2) Die Zyklomatische Zahl
 - 3) Ergebnisse
- 3) Anwendung und Erfahrung in der Realität
- 4) Vereinfachungen
 - 1) Anhand der Formel
 - 2) Anhand des Kontrollflussgraphen

2.1) Grundlagen

Voraussetzungen:

- Grundlagen aus der Graphentheorie
- Kontrollflussgraph
- Formel

Ziele:

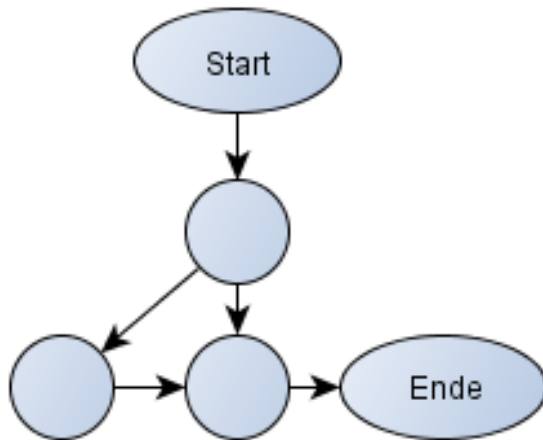
- Messen und Kontrollieren aller möglichen Pfade eines Programmablaufs

Problem ?

- Rekursiver Aufruf oder Rücksprung im Programm führt zu unendlich vielen Pfaden

2.1) Grundlagen

Kontrollflussgraph:

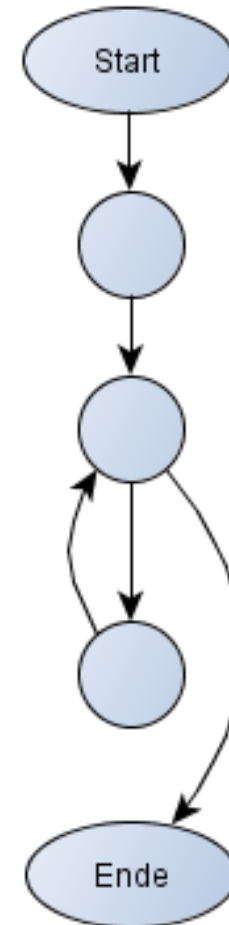


Eigenschaften:

- Jeder Knoten ist erreichbar
- Von jedem Knoten kann das Ende erreicht werden
- Die Kanten sind gerichtet
- Bildet Code ab

2.1) Grundlagen

```
private static int fac(int n){  
  
    int result = 1;  
  
    while(n > 1){  
        result *= n;  
        n--;  
    }  
    return result;  
}  
  
public static void main(String [] args){  
  
    System.out.println(fac(10));  
}
```



Gliederung Teil 1

- 1) Wichtigkeit eines Komplexitätsmaßes
- 2) Die Zyklomatische Komplexität**
 - 1) Grundlagen
 - 2) Die Zyklomatische Zahl**
 - 3) Ergebnisse
- 3) Anwendung und Erfahrung in der Realität
- 4) Vereinfachungen
 - 1) Anhand der Formel
 - 2) Anhand des Kontrollflussgraphen

2.2) Die Zyklomatische Zahl

Lösung des Kreisproblems: Rückführung auf Basispfade

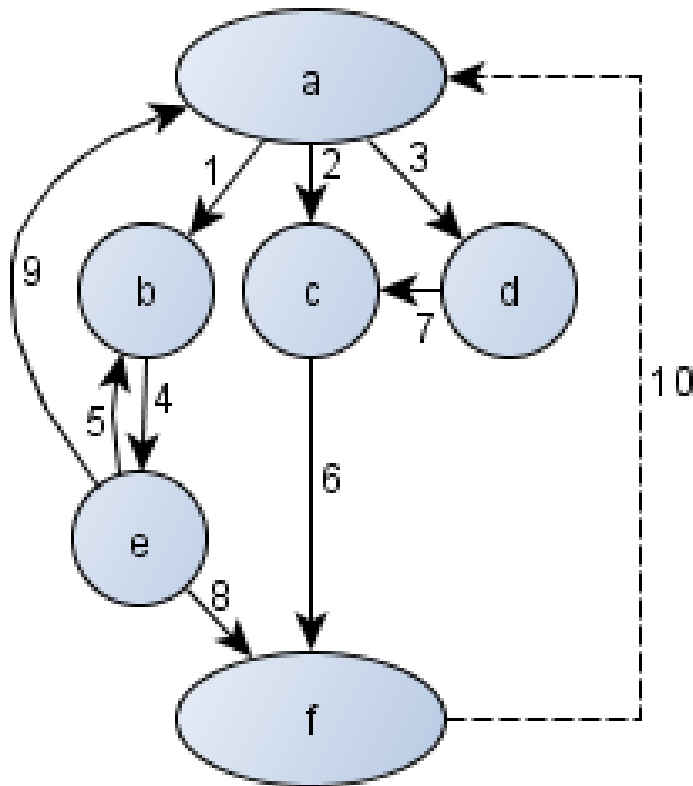
Theorem: In einem geschlossenen Graphen ist die Zyklomatische Zahl gleich der Anzahl linear unabhängiger Pfade (Basispfade).

- Geschlossener Graph:
Es gibt einen Pfad zwischen jedem Knotenpaar

2.2) Die Zyklomatische Zahl

- Lineare Unabhängigkeit:
 - „...dass sich keiner der Vektoren als Linearkombination der anderen Vektoren darstellen lässt“
- Basis:
 - *Eine Basis ist eine Teilmenge eines Vektorraumes, mit deren Hilfe sich jeder Vektor des Raumes eindeutig als endliche Linearkombination darstellen lässt.*
 - Vektoren bezeichnen in unserem Fall Pfade

2.2) Die Zyklomatische Zahl



–	1	2	3	4	5	6	7	8	9	10
$(abefa)$	1	0	0	1	0	0	0	1	0	1
(beb)	0	0	0	1	1	0	0	0	0	0
$(abea)$	1	0	0	1	0	0	0	0	1	0
$(acfa)$	0	1	0	0	0	1	0	0	0	1
$(adcfa)$	0	0	1	0	0	1	1	0	0	1

Beispiele:

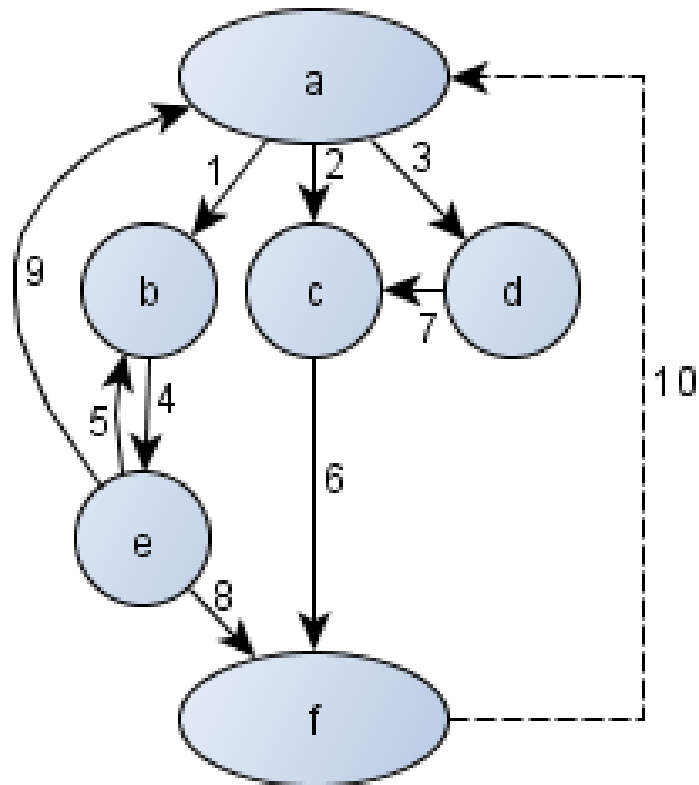
$$(adcfa) + 2(abea) + (acfa)$$

$$(abea) + 2(beb) + (abefa)$$

2.2) Die Zyklomatische Zahl

Eine Basis muss nicht
eindeutig sein, folgendes
z.B. ist auch eine Basis
des Graphen:

- (abef)
- (abeabef)
- (abebef)
- (acf)
- (adcf)



2.2) Die Zyklomatische Zahl

Die Zyklomatische Zahl:

$$v(G) = e - n + 2p$$

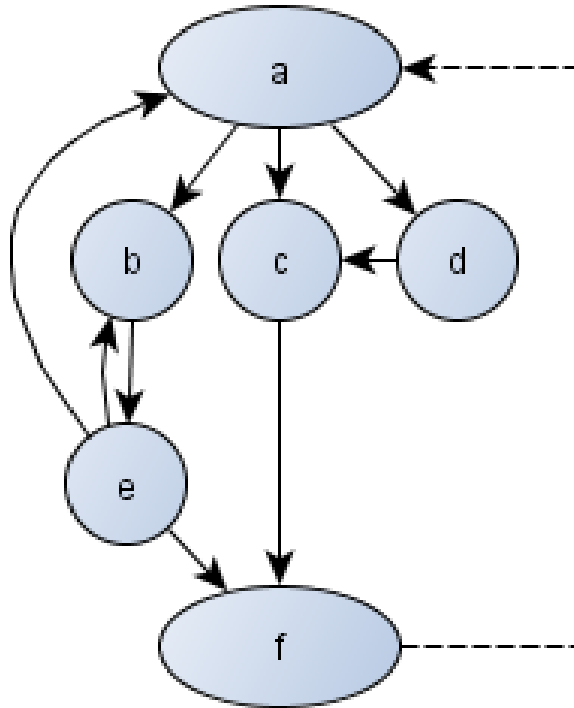
Wobei:

$e \triangleq$ Anzahl Kanten

$n \triangleq$ Anzahl Knoten

$p \triangleq$ Anzahl Komponenten

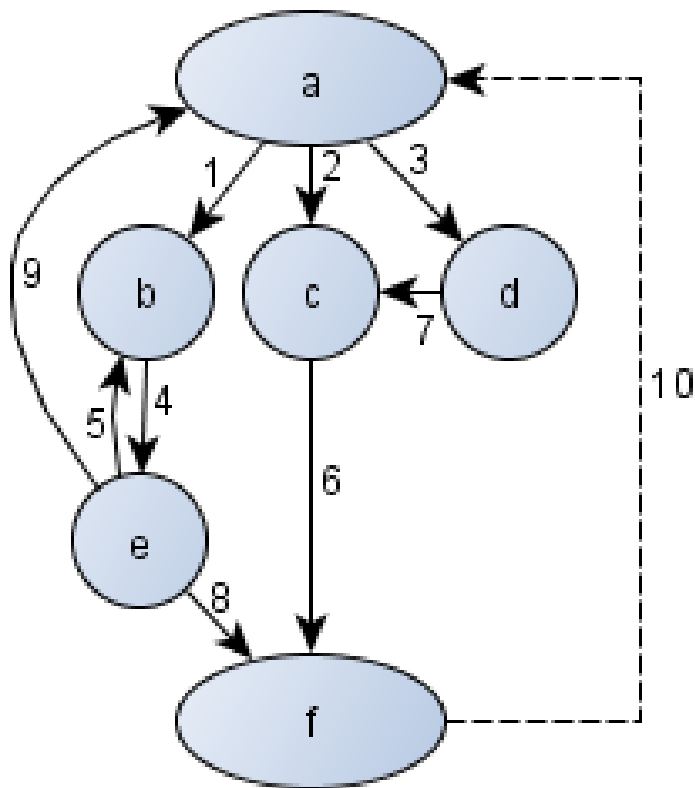
2.2) Die Zyklomatische Zahl



$$\begin{aligned}
 v(G) &= e - n + 2p \\
 &= 9 - 6 + 2 \\
 &= 5
 \end{aligned}$$

Es muss also eine Basis mit 5 Pfaden geben

2.2) Die Zyklomatische Zahl



—	1	2	3	4	5	6	7	8	9	10
<i>(abefa)</i>	1	0	0	1	0	0	0	1	0	1
<i>(beb)</i>	0	0	0	1	1	0	0	0	0	0
<i>(abea)</i>	1	0	0	1	0	0	0	0	1	0
<i>(acfa)</i>	0	1	0	0	0	1	0	0	0	1
<i>(adcfa)</i>	0	0	1	0	0	1	1	0	0	1

2.3) Zwischenergebnisse

- $v(G) \geq 1$
- $v(G)$ ist die maximale Anzahl linear unabhängiger Pfade
- Einfache Statements (z.B. Ausgaben) ändern die Komplexität nicht
- $v(G)$ hängt also nur von der Verzweigungsstruktur ab

Gliederung Teil 1

- 1) Wichtigkeit eines Komplexitätsmaßes
- 2) Die Zyklomatische Komplexität
 - 1) Grundlagen
 - 2) Die Zyklomatische Zahl
 - 3) Ergebnisse
- 3) Anwendung und Erfahrung in der Realität**
- 4) Vereinfachungen
 - 1) Anhand der Formel
 - 2) Anhand des Kontrollflussgraphen

3) Zyklomatische Komplexität in der Realität

- Automatisierte Tools, z.B. „Flow“
- Programmiert mit APL(A Programming Language, IBM)
- Spezielle Plattform PDP-10
- Schlüsselwörter sind if, goto....

```

Z←Y RUNDE X
  ⍝ Rundet Zahlen X auf Y Dezimalstellen
  ⍝ Standardwert für Y ist 2, wenn Y nicht definiert
  →(0≠⍵NC 'Y')/RN
  Y←2 ⍝ Standardwert für Y
RN:Z←(10*-Y)X|0.5+XX10*Y

```

3) Zyklomatische Komplexität in der Realität

- Visual Studio 2010:
 - integriert viele Codemetriken und Analysewerkzeuge
- Eclipse:
 - Zahlreiche Plugins wie z.B. Jmetrics oder Eclipse Metrics
 - <http://sourceforge.net/projects/jmetrics/>
 - <http://eclipse-metrics.sourceforge.net/>

```

Program.cs x Form1.Designer.cs Form1.cs [Design]
WindowsFormsApplication3.Program fac(int n)

private static int fac(int n)
{
    int result = 1;

    while (n > 1)
    {
        result *= n;
        n--;
    }
    return result;
}
}

```

Code Metrics Results

Filter: None Min: Max:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth
WindowsFormsApplication3 (Debug)	76	8	8
WindowsFormsApplication3	76	8	8
Form1	79	5	5
Program	72	5	5
fac(int) : int	71	2	2
Main() : void	76	1	1

3) Zyklomatische Komplexität in der Realität

Beobachtungen:

- Muster erkennbar (Dazu mehr in Kapitel 5)
- Zehn als guter Grenzwert (1976)
 - **Höhere Komplexität erfordert Umstrukturierung, Modularisierung**
- Heute eher geringere Grenzwerte
- Ausnahme sind Switch-Statements

3) Zyklomatische Komplexität in der Realität

```
const String wochentagsName(const int nummer) {  
    switch (nummer)  
    {  
        case 1: return "Montag";  
        case 2: return "Dienstag";  
        case 3: return "Mittwoch";  
        case 4: return "Donnerstag";  
        case 5: return "Freitag";  
        case 6: return "Samstag";  
        case 7: return "Sonntag";  
    }  
    return "(unbekannter Wochentag)";  
}
```

-> Für den menschlichen Verstand keineswegs „Komplex“

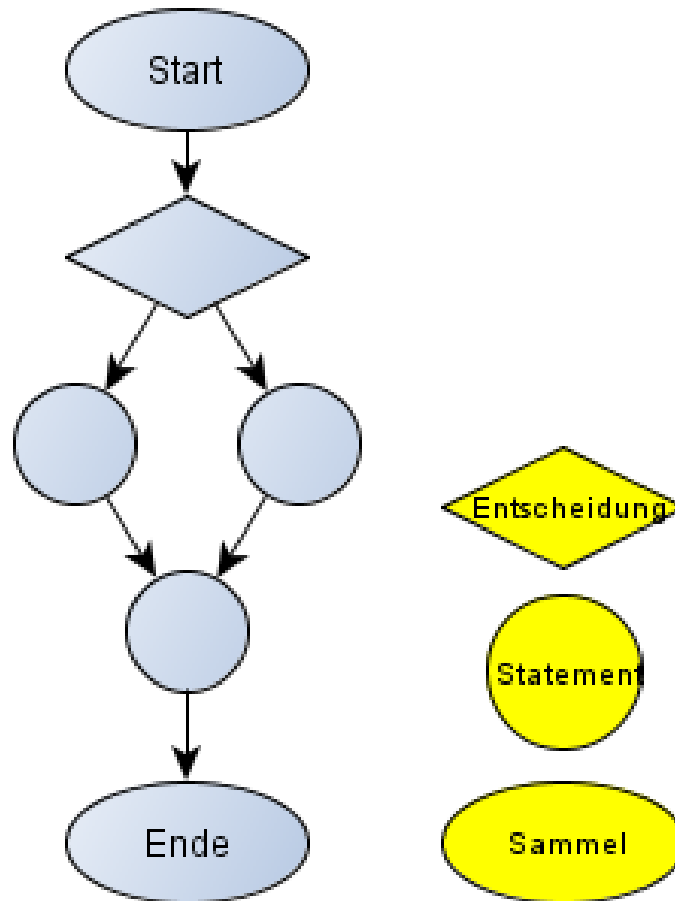
3) Zyklomatische Komplexität in der Realität

- Die Zeit, in der die Zyklomatische Komplexität eine große Bedeutung hatte, ist vorbei
- Damals war z.B. „Goto“ normaler Bestandteil einer jeden Programmiersprache
- Probleme und Konstrukte, die über die Zyklomatische Komplexität erkennbar werden, sind in modernen Programmiersprachen bereits berücksichtigt (dazu später mehr)
 - **Modularisierung !**

Gliederung Teil 1

- 1) Wichtigkeit eines Komplexitätsmaßes
- 2) Die Zyklomatische Komplexität
 - 1) Grundlagen
 - 2) Die Zyklomatische Zahl
 - 3) Ergebnisse
- 3) Anwendung und Erfahrung in der Realität
- 4) Vereinfachungen**
 - 1) Anhand der Formel**
 - 2) Anhand des Kontrollflussgraphen

4.1 Vereinfachung der Berechnung anhand der Formel



4.1 Vereinfachung der Berechnung anhand der Formel

$|Funktionsknoten| = \theta$ (Statements)

$|Entscheidungsknoten| = \pi$

$|Sammelknoten| = \gamma$

Es gilt : $n = \theta + 2\pi + 2$, weil

- es genau einen Eingangs und Ausgangsknoten gibt
- genau 2 Knoten bei einer Entscheidung entstehen (Sammel- und Entscheidungsknoten)

4.1 Vereinfachung der Berechnung anhand der Formel

Ausgangsformel aus letzter Folie: $n = \theta + 2\pi + 2$

Es gilt: $e = 1 + \theta + 3\pi$

Einsetzen in $v = e - n + 2$ ergibt

$$\begin{aligned} v &= (1 + \theta + 3\pi) - (\theta + 2\pi + 2) + 2 \\ &= \pi + 1 \end{aligned}$$

4.1 Vereinfachung der Berechnung anhand der Formel

Beispiel:

$$v(G) = \pi + 1 = 3 + 1 = 4$$

Zur Kontrolle:

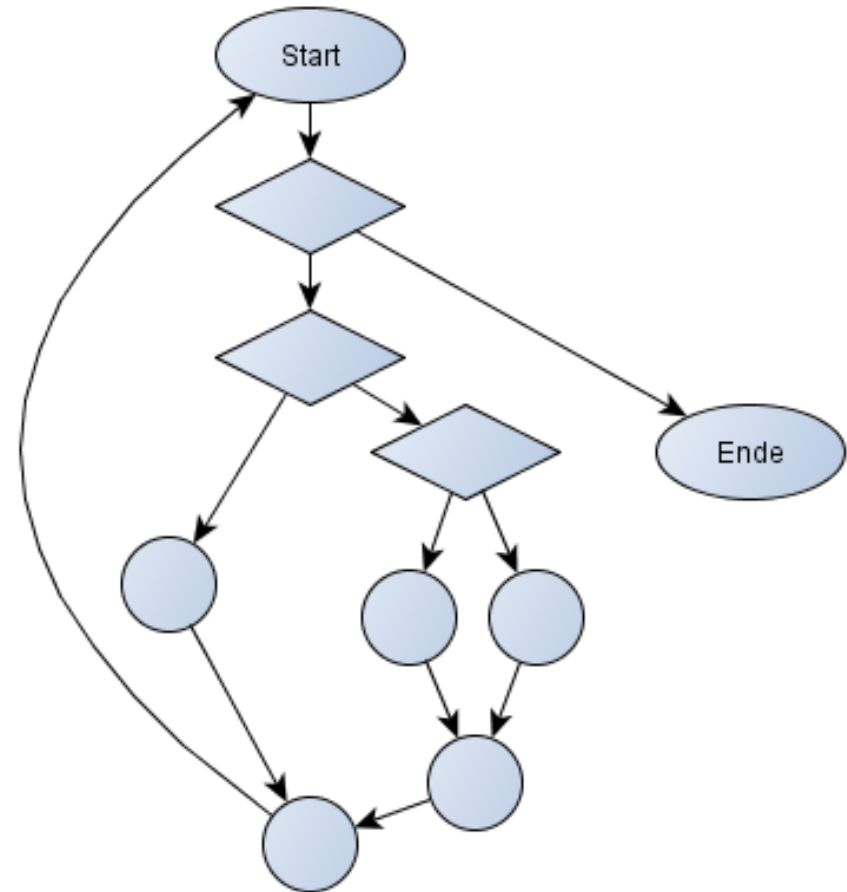
$$\begin{aligned} v(G) &= e - n + 2p \\ &= 12 - 10 + 2 \\ &= 4 \end{aligned}$$

Zur Erinnerung

e = Kanten

n = Knoten

p = Blöcke



Gliederung Teil 1

- 1) Wichtigkeit eines Komplexitätsmaßes
- 2) Die Zyklomatische Komplexität
 - 1) Grundlagen
 - 2) Die Zyklomatische Zahl
 - 3) Ergebnisse
- 3) Anwendung und Erfahrung in der Realität
- 4) Vereinfachungen**
 - 1) Anhand der Formel
 - 2) Anhand des Kontrollflussgraphen**

4.2 Vereinfachung der Berechnung anhand des Graphen

Basis: Eulerscher Polyedersatz

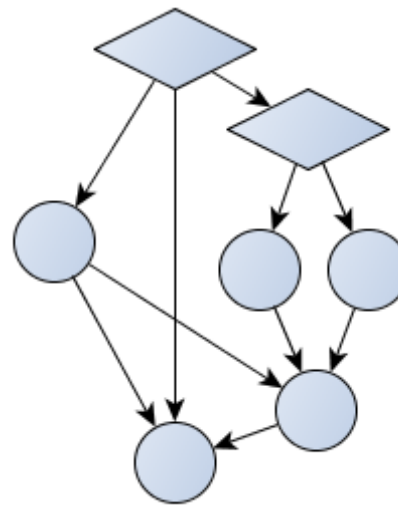
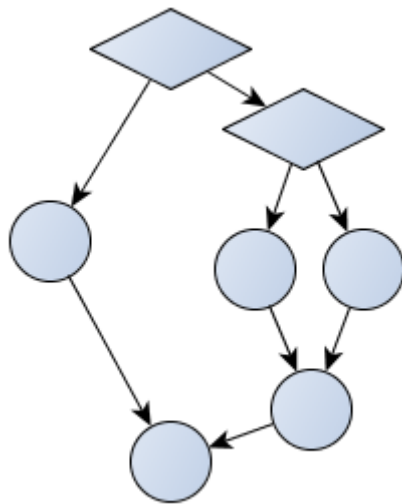
Bei **planaren** Graphen gilt:

$$n - e + r = 2$$

wobei r für die Regionen des Graphen steht

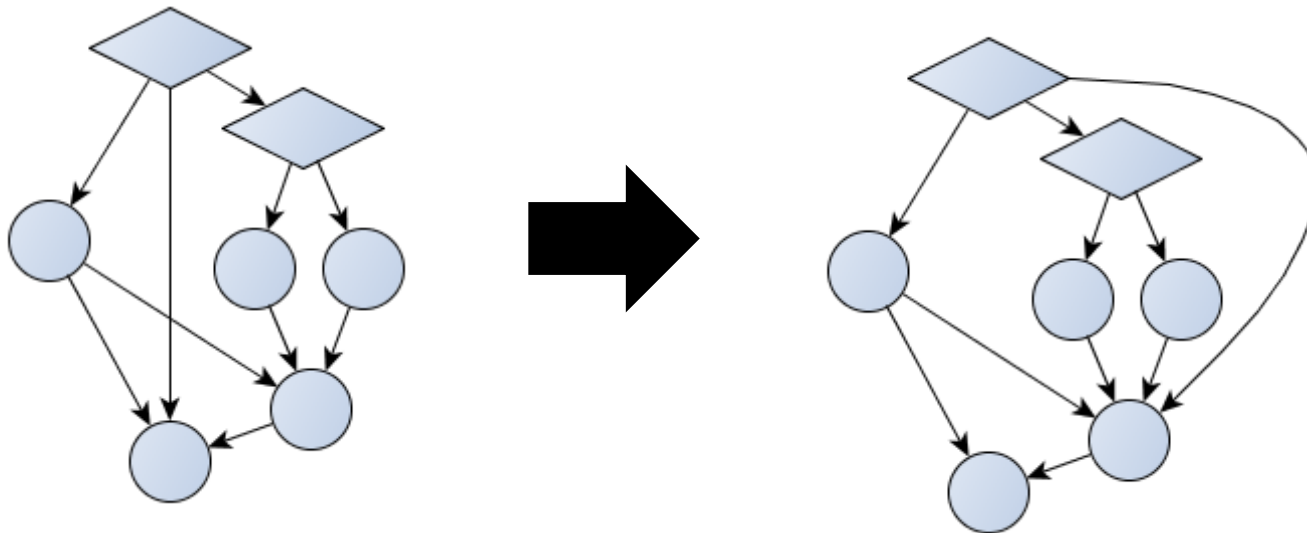
4.2 Vereinfachung der Berechnung anhand des Graphen

Erinnerung: Planare Graphen



Wirklich ?

4.2 Vereinfachung der Berechnung anhand des Graphen



Problem: Graph muss korrekt umgeformt sein

4.2 Vereinfachung der Berechnung anhand des Graphen

Basis: Eulerscher Polyedersatz

Bei **planaren** Graphen gilt:

$$n - e + r = 2$$



$$r = e - n + 2$$

r ist die Anzahl der Regionen

Letzeres entspricht der zyklomatischen
Komplexität

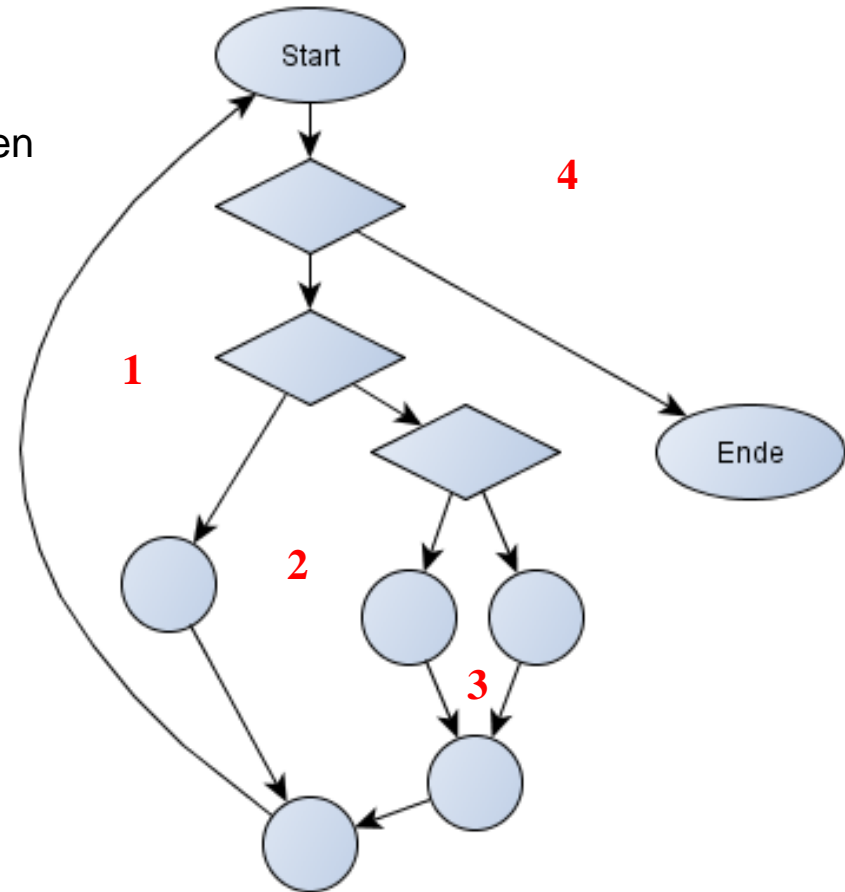
4.2 Vereinfachung der Berechnung anhand des Graphen

Beispiel:

- Im gezeigten Graphen gibt es 4 Regionen

Zur Kontrolle:

$$\begin{aligned}
 v(G) &= e - n + 2p \\
 &= 12 - 10 + 2 \\
 &= 4
 \end{aligned}$$



Gliederung Teil 2

- 5) Strukturlose Programmierung**
- 6) Testmethodik

5) Strukturlose Programmierung

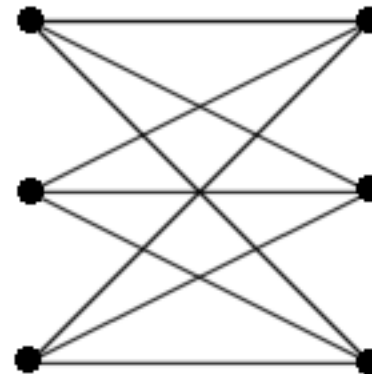
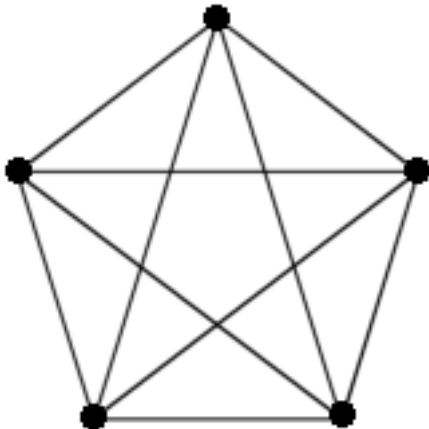
- Aus der Praxis zeigt sich dass es auch Situationen gibt in denen „unstrukturierter“ Code von Vorteil ist (vgl. Switch Statements)
- Woran erkennt man überhaupt unstrukturierten Code ?

 **Pendant in der Graphentheorie**

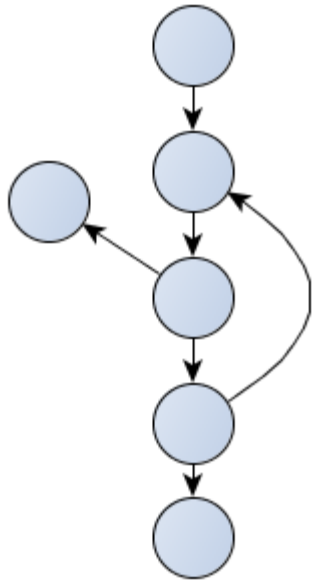
5) Strukturlose Programmierung

Der Satz von Kuratowski: Mustergraphen

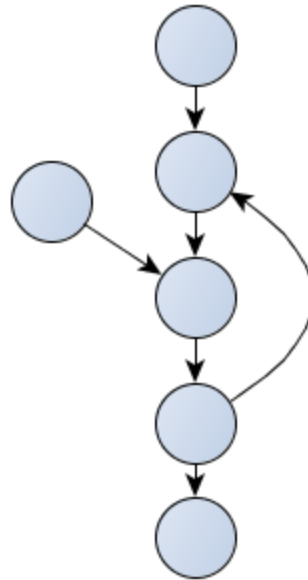
- Jeder nicht-planare Graph enthält mindestens einen von 2 speziellen Graphen, nämlich dem K_5 oder $K_{3,3}$



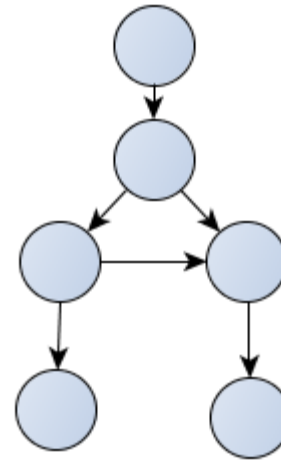
5) Strukturlose Programmierung



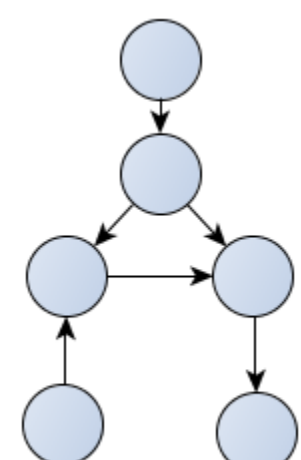
**Aus Schleife
herausspringen**



**In Schleife
hineinspringen**



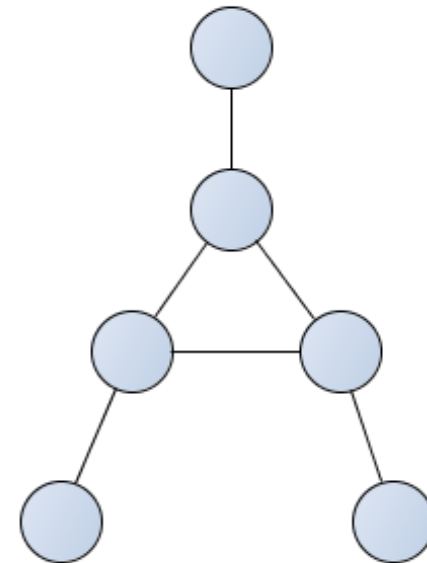
**Aus
Entscheidung
herausspringen**



**In Entscheidung
hineinspringen**

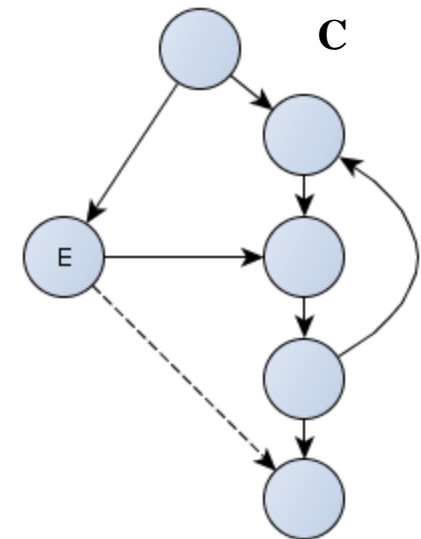
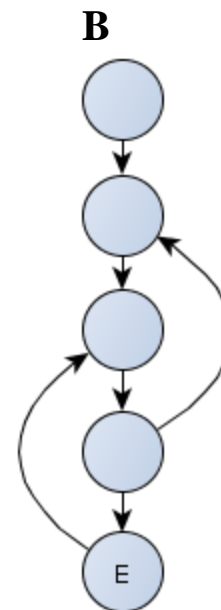
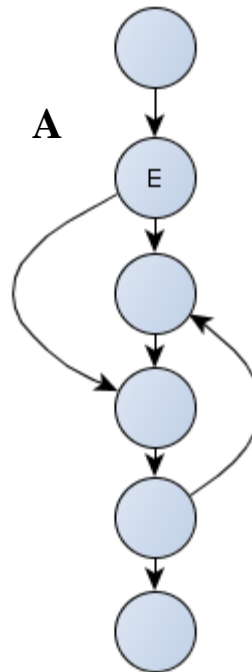
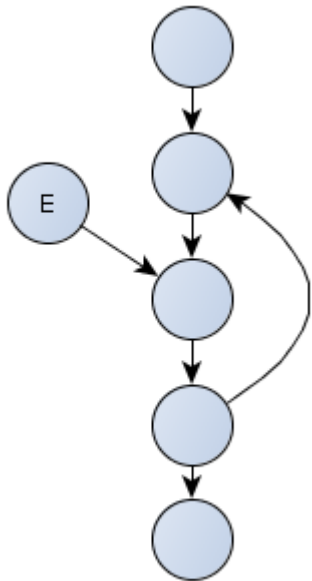
5) Strukturlose Programmierung

- ❖ **Jedes Programm, das 2 dieser Muster enthält, gilt als unstrukturiert**
- ❖ **Die Graphen können nicht alleine auftreten (siehe nächste Folien)**
- ❖ **Alle Mustergraphen sind isomorph zum gezeigten ungerichteten Graph**
- ❖ **Solche Muster sind mit heutigen Programmierstrukturen nicht mehr möglich**



5) Strukturlose Programmierung

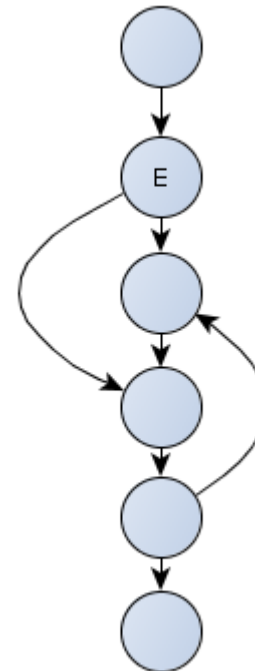
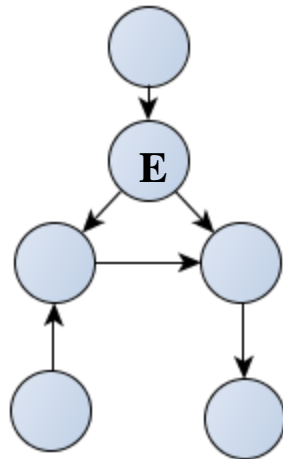
3 Möglichkeiten:



5) Strukturlose Programmierung

Möglichkeit A:

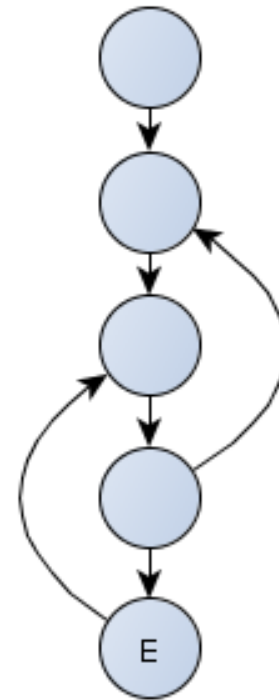
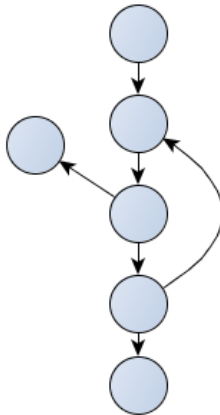
- Folgender Graph ist dann zwingend auch enthalten:



5) Strukturlose Programmierung

Möglichkeit B:

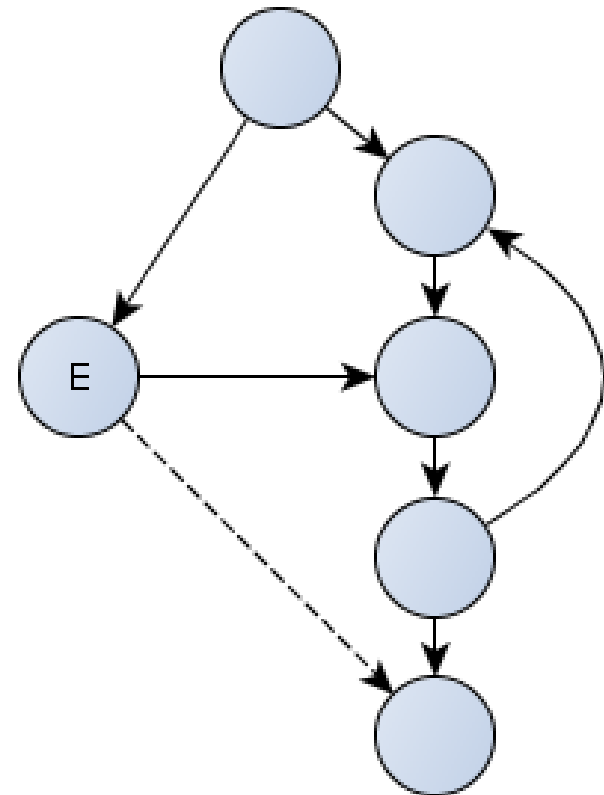
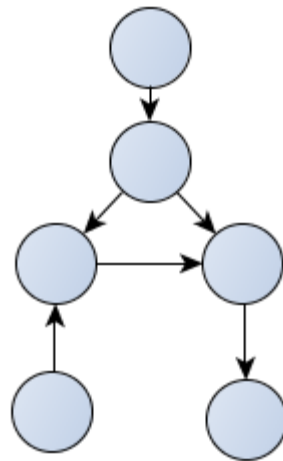
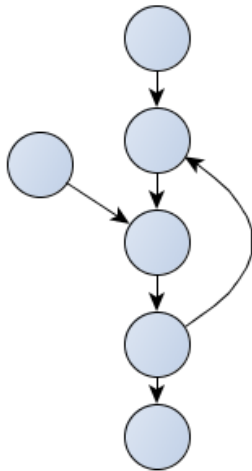
- Folgender Graph ist dann zwingend auch enthalten:



5) Strukturlose Programmierung

Möglichkeit C:

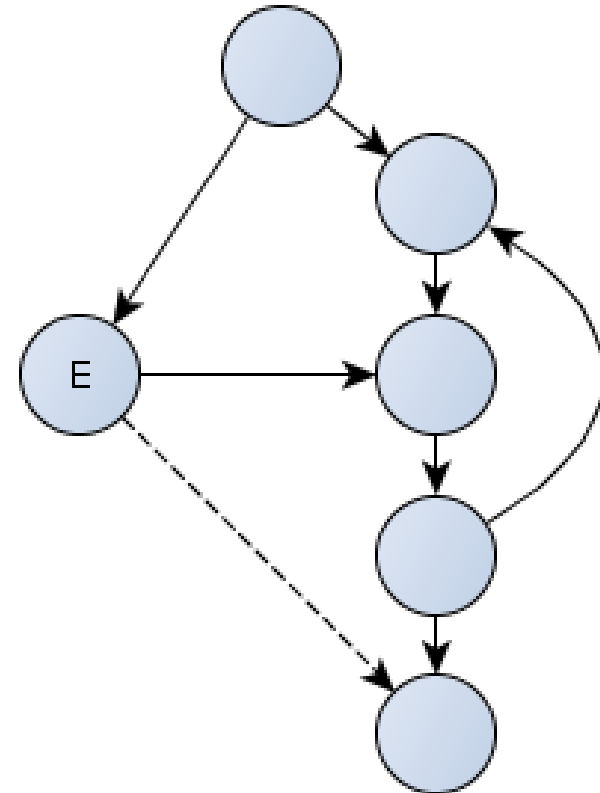
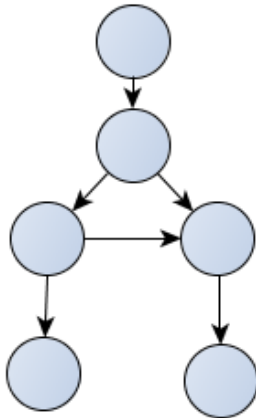
- Folgende Graphen sind dann auch enthalten:



5) Strukturlose Programmierung

Möglichkeit C:

- Folgender Graphen ist enthalten, wenn es eine Kante von E zu einem anderen Knoten als in die Schleife gibt:

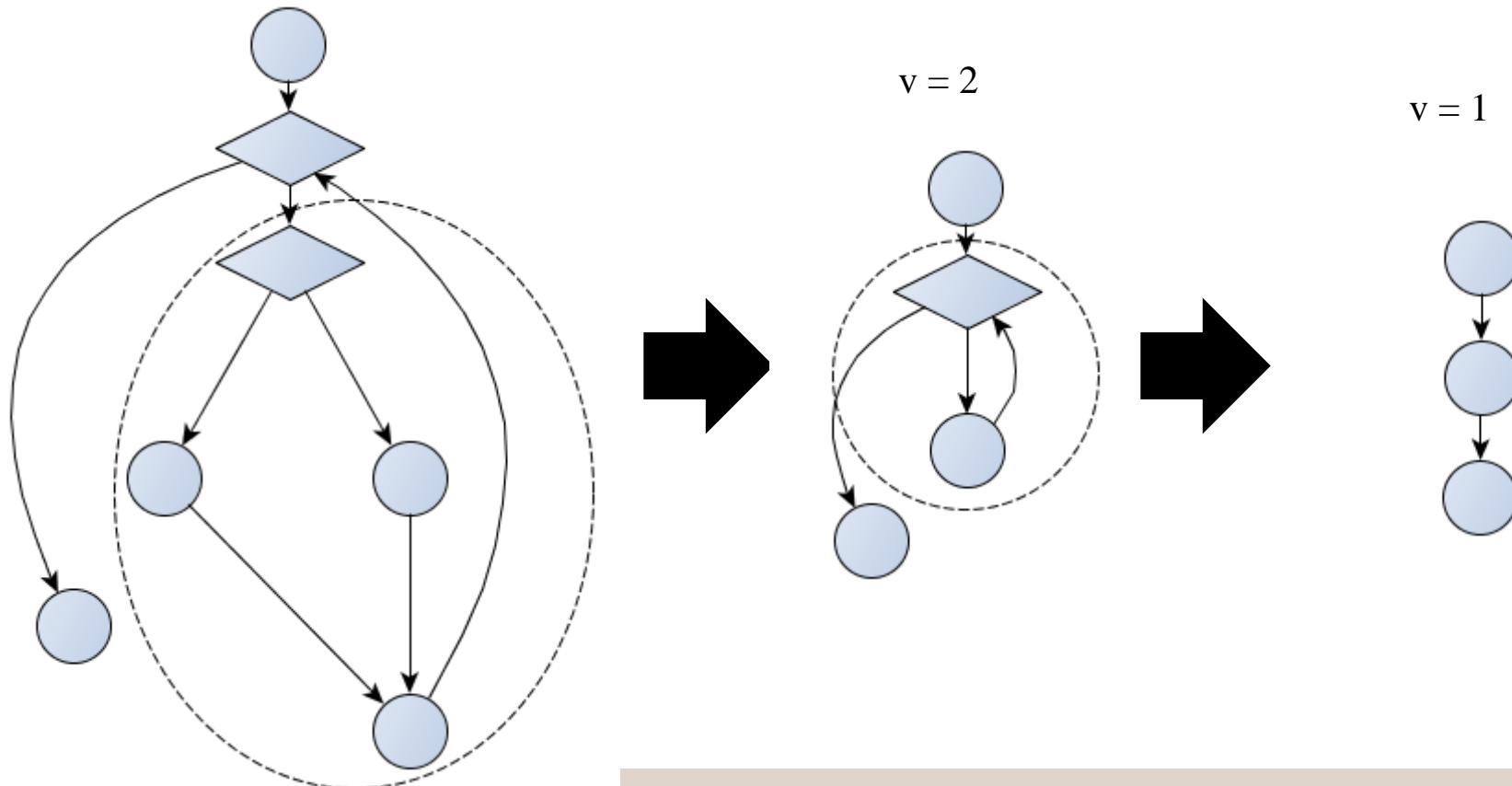


5) Strukturlose Programmierung

- Ein weiteres Merkmal Strukturloser Programme ist, dass der Kontrollflussgraph nicht vereinfacht werden kann
- Grund hierfür ist das Fehlen von eindeutigen Start- und Endknoten des Graphen oder Subgraphen (Notwendigkeit um Modularisierung durchzuführen)
- In einem strukturierten Programm kann die Struktur soweit vereinfacht werden, dass die Zyklomatische Komplexität eins beträgt

5) Strukturlose Programmierung

Ein Beispiel:



5) Strukturlose Programmierung

Formal:

Essentielle Komplexität ev :

$$ev = v - n$$

- Wenn diese kleiner oder gleich der Zyklomatischen Komplexität ist, kann der Graph vereinfacht werden
- Ein strukturiertes Programm hat Essentielle Komplexität von eins

Gliederung Teil 2

- 5) Strukturlose Programmierung
- 6) Testmethodik**

6) Testmethodik

- Zyklomatische Komplexität gibt Anhaltspunkt für Komplexität
- Beim Testen von Programmen kann auf die Zyklomatische Komplexität zurückgegriffen werden
- Sie kann dabei sogar sehr nützlich sein, um den Testaufwand zu verringern
- Geringere Zyklomatische Komplexität bedeutet geringeren Testaufwand

6) Testmethodik

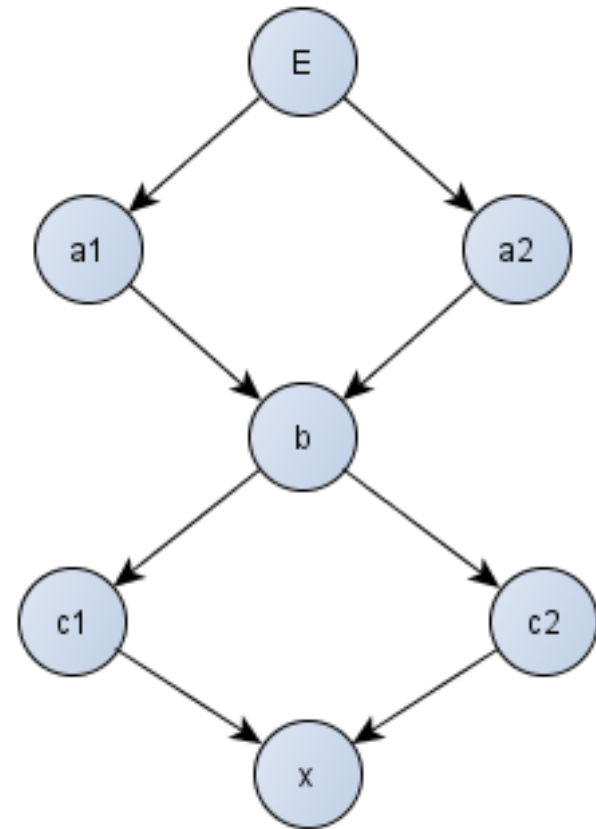
Ausgangspunkt:

- Ein Programm P mit Komplexität v
- Die Zahl der getesteten Pfade ist **ac** (actual complexity)
- Wenn **ac** $<$ **v** ist muss eine der drei folgenden Bedingungen greifen:
 - 1) Es müssen mehr Pfade getestet werden
 - 2) Der Kontrollflussgraph kann vereinfacht werden
($v - ac$ Entscheidungen können eingespart werden)
 - 3) Bestimmte Verzweigungen sind voneinander abhängig und können zusammengefasst werden

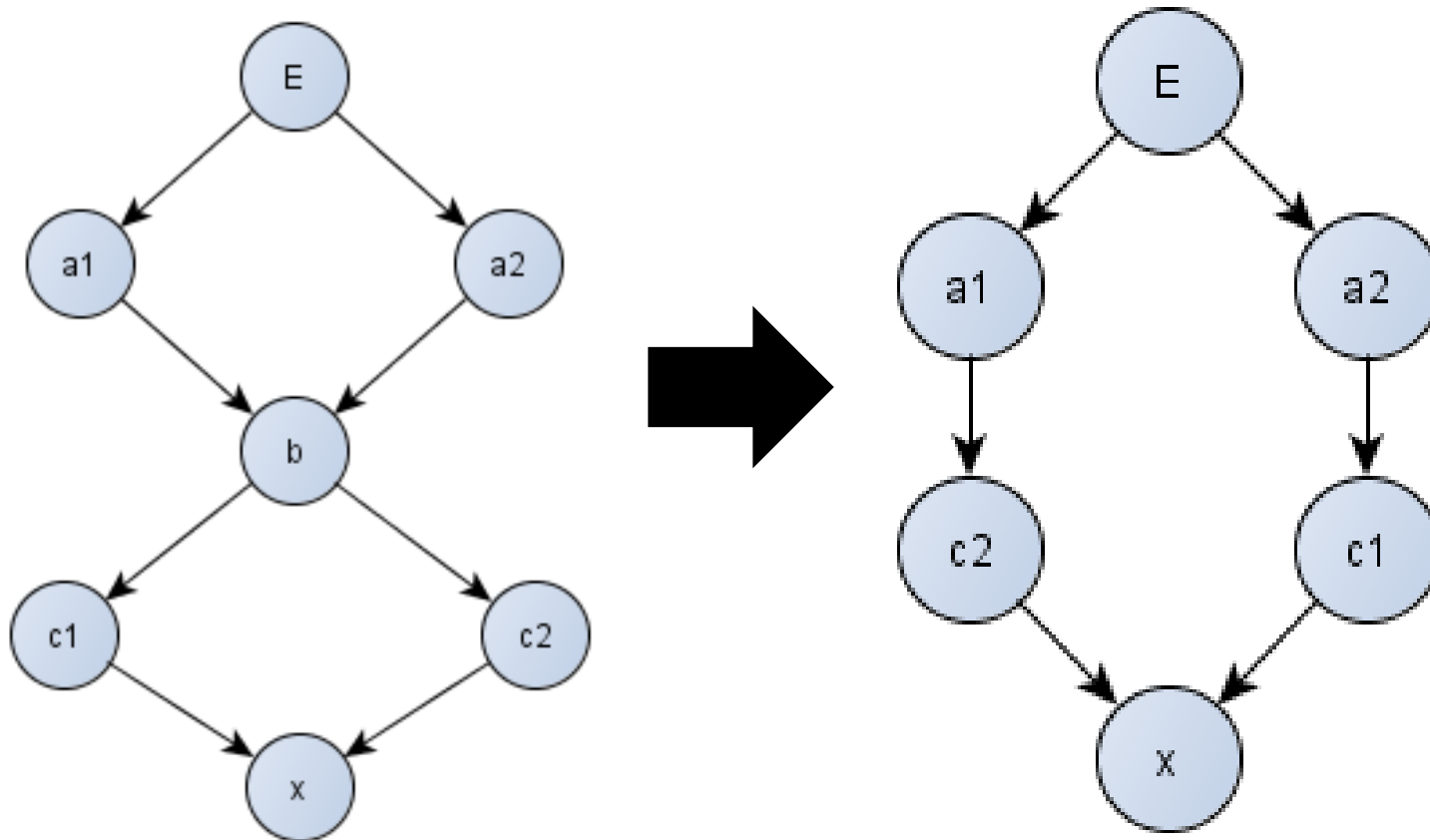
6) Testmethodik

Beispiel:

- $ac = 2$, die 2 getesteten Pfade sind $[E, a1, b, c2, x]$ und $[E, a2, b, c1, x]$
- $[E, a1, b, c1]$ und $[E, a2, b, c2]$ können nicht ausgeführt werden
- Der Graph kann offensichtlich vereinfacht werden



6) Testmethodik



**Vielen Dank für die
Aufmerksamkeit**