Codekomplexität

- Die LOC sagen wenig über die Komplexität der Software aus, weil die Beziehungen (Abfragen, Aufrufe, Sprünge, ...) zwischen den Anweisungen und Daten nicht berücksichtigt werden
- Einer der ersten Vorschläge für eine Metrik zur Messung der Codekomplexität stammt von Halstead

- Halstead-Metriken
  - Orientieren sich an dem Komplexitätsbegriff aus der Informationstheorie nach Shannon
    - Eine komplexe Nachricht enthält viele unterschiedliche Zeichen relativ zur Länge der Nachricht
  - Ein Quellcode ist nach Halstead komplex, wenn in den Befehlen viele unterschiedliche Operatoren und Operanden relativ zur Anzahl der Befehle vorkommen

- Halstead teilt die Programmelemente in Operatoren und Operanden ein
  - Operatoren
    - Schlüsselwörter
    - Operatoren
    - Präprozessoranweisungen
  - Operanden
    - Variablen
    - Konstanten
    - Bezeichner

- Für die Berechnung der Metriken werden die folgenden Basisparameter benötigt
  - t : Anzahl der unterschiedlichen Operatoren
  - d : Anzahl der unterschiedlichen Operanden
  - n_t : Gesamtzahl aller Operatoren im Programm
  - n_d : Gesamtzahl aller Operanden im Programm

können über Syntaxanalyse ermittelt werden

– Beispiel

```
int ggt(int a, int b){
   while(a != b){
      if (a > b)
         a -= b;
      else
         b -= a;
   }
   return a;
}
```

$t = 11$

$d = 2$

$n_t = 20$

$n_d = 11$

Operatoren

`int`	3 x	`(), {}`	5 x
`,`	1 x	`while`	1 x
`!=`	1 x	`if else`	1 x
`>`	1 x	`-=`	2 x
`return`	1 x	`;`	3 x
`ggt`	1 x		

Operanden

`a`	6 x	`b`	5 x

- Aus den Basisgrößen werden weitere Größen berechnet
  - Größe des Vokabulars: $G = t + d$
  - Länge des Programms: $N = n_t + n_d$
  - Volumen des Programms: $V = N \times \log_2 G$

Annahme: Operatoren und Operanden werden mit der gleichen Anzahl von Bits codiert

- Die Komplexität berechnet Halstead in Bezug auf eine minimale Implementierung (G und N minimal)
- Da zu einem beliebigen Programm die Angabe einer minimalen Implementierung nicht trivial ist, arbeitet Halstead mit unteren Schranken *d.h. keine Implementierung kann ein kleineres Volumen haben*
  - Eine untere Schranke für die Operanden ist die Anzahl der Ein- und Ausgabeoperanden: d^*
  - Wenn eine (fiktive) Programmiersprache einen speziellen Operator ◆ hat, welcher das komplette Programm berechnet, dann würde das Programm nur aus einer Zeile der folgenden Form bestehen

$$\texttt{Operand}_1 = \blacklozenge(\texttt{Operand}_2, \ldots, \texttt{Operand}_{d^*})$$

- Die Anzahl der Operatoren würde für dieses Programm 2 (◆, =) betragen
- Damit gilt für die Größe des Minimalvokabulars: $G^* = d^* + 2$
- Entsprechend wir ein Minimalvolumen des Programms definiert: $V^* = G^* \times \log_2 G^*$

Klammerung wird hier nicht gezählt (abhängig von der Programmiersprache)

– Aufgabe: Berechnen Sie G, N, d^*, t^*, G^*, V und V^* für das Beispielprogramm `ggt`

– Der Level L misst, zu welchem Grad die (reale) Implementierung von der Minimalimplementierung abweicht:

$$L = V^* / V$$

im „optimalen" Fall ist $L = 1$

soll eine Aussage über die Transparenz des Programmes liefern

- Ein Programm ist umso einfacher aufgebaut, je näher sich *L* der Zahl 1 nähert
- Daraus ergibt sich die Schwierigkeit (*difficulty*) eines Programms

$$D = 1 / L$$

- Der Aufwand (*effort*), der zum Erstellen bzw. Verstehen des Programms benötigt wird, ergibt sich nach Halstead als das Produkt von Programmvolumen und Schwierigkeitsgrad

$$E = V \times D$$

- Der Aufwand, der für ein Programmmodul verwendet werden muss, steigt damit überproportional mit dem Code-Volumen an

$$E = V \times D = V / L = V / (V^* / V) = V^2 / V^*$$

- Aufgabe: Berechnen Sie *L*, *D* und *E* für das Beispielprogramm `ggt`

- Kritik an Halstead-Metriken
  - Einteilung in Operatoren und Operanden nicht immer klar (z.B. Lisp)
  - Die Metriken beziehen sich allein auf syntaktische Eigenschaften des Programms. Semantische Eigenschaften, wie z.B. der Kontrollfluss, werden nicht berücksichtigt

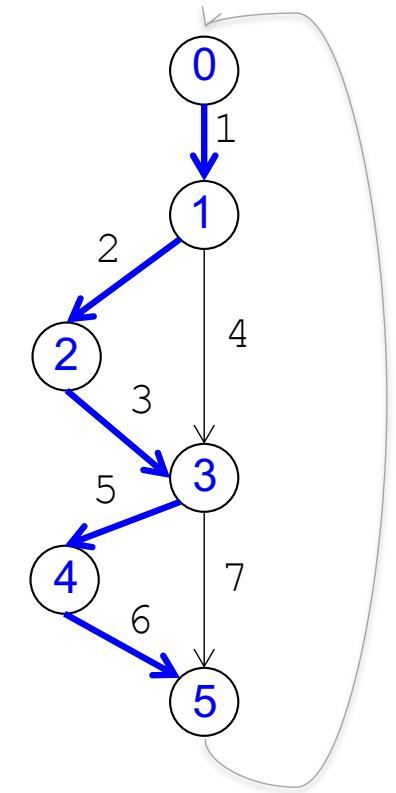
- McCabe-Metrik
  - Anstatt syntaktische Elemente zu betrachten (wie Halstead), konzentriert sich McCabe auf den Kontrollfluss eines Programms
  - Der Programmcode wird dazu in einen Graphen überführt (Kontrollflussgraph)
  - Mit Hilfe der Graphentheorie nach Euler werden dann Aussagen über den Kontrollflussgraphen getroffen
  - Sei $G=(N,E)$ ein Graph mit der Knotenmenge $N=\{1,...,|N|\}$ und der Kantenmenge $E=\{1,...,|E|\}$
  - Ein Pfad kann, wenn man von der Reihenfolge der besuchten Knoten abstrahiert, als Vektor $p=(p_1,...,p_{|E|}) \in \mathbb{N}^{|E|}$ dargestellt werden
  - p_i gibt dabei an, wie häufig der Pfad die i-te Kante des Graphen durchläuft
  - Man kann zeigen, dass sich jeder geschlossene Pfad (Start- und Endknoten sind gleich) als Linearkombination von Elementarpfaden darstellen kann
  - Die minimale Menge von Elementarpfaden wird als Basis bezeichnet

– Beispiel (aus [Hof13]):

```
0:  public static int manhatten(int a, int b){
1:     if (a < 0)
2:        a = -a;
3:     if (b < 0)
4:        b = -b;
5:     return a+b;
    }
```

$$\begin{pmatrix}1\\1\\1\\0\\1\\1\\0\end{pmatrix} = 1\cdot\begin{pmatrix}1\\1\\1\\0\\0\\0\\1\end{pmatrix} + 1\cdot\begin{pmatrix}1\\0\\0\\1\\1\\1\\0\end{pmatrix} - 1\cdot\begin{pmatrix}1\\0\\0\\1\\0\\0\\1\end{pmatrix}$$

Hilfskante wurde hier ausgeblendet



Hilfskante, um einen geschlossenen Pfad zu bekommen

- Die zyklomatische Zahl

hier: stark zusammenhängender Graph

$$V(G) = |E| - |N| + 1$$

Graphentheorie

gibt die Größe einer Basis für einen stark zusammenhängenden Graphen $G = (N, E)$ an

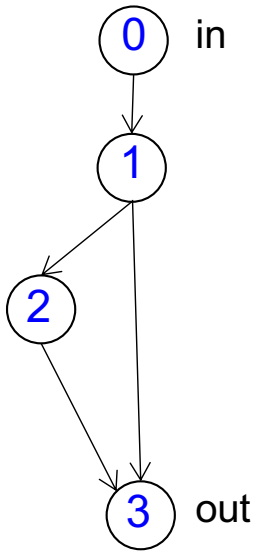
- Da ein Kontrollflussgraph durch eine Hilfskante erweitert werden muss, um einen stark zusammenhängenden Graphen zu erhalten, definiert McCabe die zyklomatische Komplexität eines Kontrollflussgraphen G durch

$$V(G) = |E| - |N| + 2$$

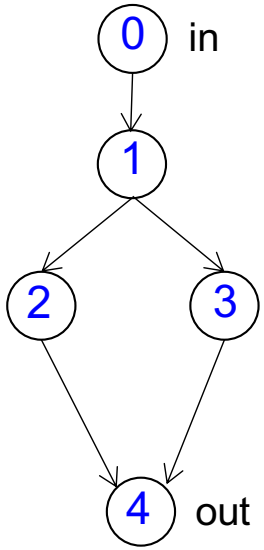
hier: Kontrollflussgraph

– Aufgabe: Berechnen Sie die zyklomatische Komplexität der beiden folgenden Abfragen

```
if (B) X;
```




```
if (B) X; else Y;
```



- Eigenschaften der zyklomatischen Komplexität
  - $V(G)$ ist unabhängig von der Programmlänge, da sequentiell durchlaufende Programmabschnitte beliebig verlängert werden können, ohne $V(G)$ zu verändern
  - $V(G)$ wird nur durch die Struktur des Kontrollflussgraphen beeinflusst, die Zusammenfassung von Befehlen spielt keine Rolle
  - Wird zum Kontrollflussgraphen eine einzelne Kante hinzugefügt, so erhöht sich $V(G)$ um 1
  - Wird aus dem Kontrollflussgraphen eine einzelne Kante entfernt, so verkleinert sich $V(G)$ um 1

- Empfehlung von McCabe: Die zyklomatische Komplexität eines Moduls sollte den Wert 10 nicht überschreiten

- Besteht ein Programm aus *n* unabhängigen Funktionen, dann ist der Kontrollflussgraph $G=(N, E)$ ein Wald aus Untergraphen $G_1,...,G_n$.
- Dann gilt eine verallgemeinerte Form der zyklomatischen Komplexität

$$v(G) = |E| - |N| + 2n = \sum_{i=1}^{n} V(G_i)$$

- Objektorientierte Metriken
  - Die einzelnen Methoden können mit den bekannten Metriken vermessen werden
  - Die folgenden Metriken bieten eine objektorientierte Sicht

Abkürzung	**Metrik**	**Typ**
OV	Object Variables	Umfangsmetrik
CV	Class Variables	Umfangsmetrik
NOA	Number of Attributes	Umfangsmetrik
WAC	Weighted Attribute per Class	Umfangsmetrik
WMC	Weighted Method per Class	Umfangsmetrik
DOI	Depth Of Inheritance	Vererbungsmetrik
NOD	Number of Descandants	Vererbungsmetrik
NORM	Number of Redefined Methods	Vererbungsmetrik

- Die Gewichtung bei WMC erfolgt häufig über die zyklomatische Komplexität
- Wenn A die Menge aller Attribute und M die Menge aller Methoden ist, ergibt sich damit die folgende Berechnung

$$WAC = \sum_{i=1}^{|A|} v(a_i)$$

Komplexität eines Attributs

$$WMC = \sum_{i=1}^{|M|} v(m_i)$$

Komplexität einer Methode