Softwaremetriken

Christian Silberbauer

Einführung

- Softwaremetriken bestimmen die Komplexität von Programmen.

- Beispiele:
  - Lines of Code (LOC)
  - Zyklomatische Komplexität
  - Halstead-Metrik

Lines of Code

- Anzahl der Codezeilen werden gezählt.
- Unterschieden werden:
  - PLOCs: Physical Lines Of Code
  - SLOCs: Source Lines Of Code
- Also: ohne bzw. mit Kommentar- und Leerzeilen.
- Aber:
  - Nicht jede Anweisung ist gleich komplex.
  - In einer Codezeile können mehrere Anweisungen stehen.

Zyklomatische Komplexität

- Auch bekannt als McCabe-Metrik.
- Die Zyklomatische Komplexität v eines Graphen G mit e Kanten, n Knoten und p verbundenen Komponenten wird wie folgt berechnet (unter Berücksichtigung eines Ausgangsknotens):

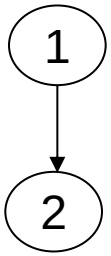
$$v(G) = e - n + 2p$$

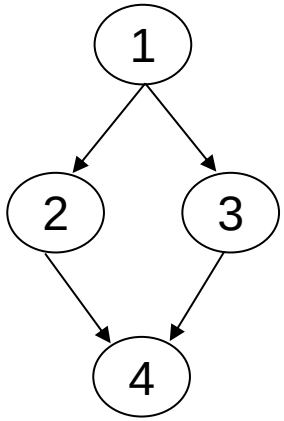
- Oder unter Berücksichtigung der Anzahl Verzweigungen b, falls es nur einen Ausgangsknoten gibt:

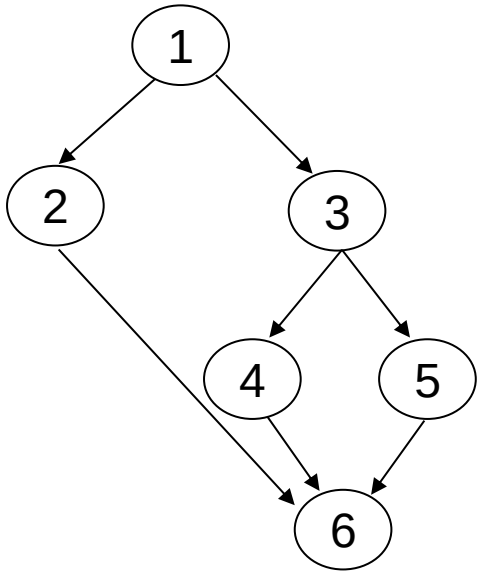
$$v(G) = b + p$$

- Damit wird die Anzahl unabhängiger Pfade im Code bestimmt.

Beispielhafte Kontrollflüsse







v(G):	1	2	3
Kontrollflüsse:	1 - 2	1 - 2 1 - 3	1 - 2 1 - 3 - 4 1 - 3 - 5

Beispielcode

```
public static int ggt1(int x, int y) {
   while (y != 0) {
      int h = y;
      y = x % y;
      x = h;
   }
   return x;
}
```

```
public static int ggt2(int x, int y) {
if (y == 0) return x;
else        return ggt2(y, x % y);
}
```

```
public static int ggt3(
      int x, int y) {

   while (x != y) {
      if (x > y) {
         x = x - y;
      } else {
         y = y - x;
      }
   }
   return x;
}
```

Beispielcode

```
public static int fib(int x) {
    if (x == 0 || x == 1) return x;
    return fib(x - 1) + fib(x - 2);
}
```

- Für die Zyklomatische Komplexität ist jeder elementare logische Ausdruck als Entscheidungspunkt zu berücksichtigen.

Beispielcode

```
static String getWochentag(int id) {
   switch (id) {
      case 1:  return "Monatag";
      case 2:  return "Dienstag";
      case 3:  return "Mittwoch";
      case 4:  return "Donnerstag";
      case 5:  return "Freitag";
      case 6:  return "Samstag";
      default: return "Sonntag";
   }
}
```

Aus McCabe, 1976, „A Complexity Measure“

“Abstract - This paper describes a graph-theoretic complexity measure and illustrates how it can be used to manage and control program complexity.”

“There is a critical question facing software engineering today: How to modularize a software system so the resulting modules are both testable and maintainable? That the issues of testability and maintainability are important is borne out by the fact that we often spend half of the development time in testing [2] and can spend most of our dollars maintaining systems [3].”

Aus McCabe, 1976, „A Complexity Measure“

“Programmers have been required to calculate complexity as they create software modules. When the complexity exceeded 10 they had to either recognize and modularize subfunctions or redo the software. The intention was to keep the "size" of the modules manageable and allow for testing all the independent paths […] The only situation in which this limit has seemed unreasonable is when a large number of independent cases followed a selection function (a large case statement), which was allowed.”

Bewertung

Halstead-Metriken

- Programmlänge: $N = N_1 + N_2$
- Größe des Vokabulars: $\eta = \eta_2 + \eta_2$
- Volumen des Programms: $V = \mathrm{N} * log_2(\eta)$
- Parameter:
  - η_1 = Anzahl verschiedener Operatoren
  - η_2 = Anzahl verschiedener Operanden
  - N_1 = Anzahl insgesamt verwendeter Operatoren
  - N_2 = Anzahl insgesamt verwendeter Operanden

Halstead-Metriken

- Programm-Level:

$$L = \frac{V^*}{V}$$

- Schwierigkeit:

$$D = \frac{1}{L}$$

- Parameter:
  - V^* = Potentielles Volumen
  - V = Volumen

Halstead-Metriken

- Dies ist laut Halstead eine alternative Formel zur Bestimmung des Programm-Levels:

$$\hat{L} = \frac{2}{\eta_1} * \frac{\eta_2}{N_2}$$

- Demnach gilt auch:

$$\hat{D} = \frac{\eta_1}{2} * \frac{N_2}{\eta_2}$$

- Parameter:
  - η_1 = Anzahl verschiedener Operatoren
  - η_2 = Anzahl verschiedener Operanden
  - N_1 = Anzahl insgesamt verwendeter Operatoren
  - N_2 = Anzahl insgesamt verwendeter Operanden

Halstead-Metriken

- (Halstead, 1977, S. 27):

$$\widehat{D} = \frac{\eta_1}{2} * \frac{N_2}{\eta_2}$$

"With respect to operators, it seems reasonable to assume that the greater the number of unique operators employed in a given implementation, the lower the level of that implementation. Now the least possible number of unique operators that could ever be employed is two, where the pair would consist of a function designator and an assignment or grouping operator."

"Operands, on the other hand, exhibit no unique minimum over all possible algorithms, so they must be treated differently. In their case, it may suffice to note that whenever the name of an operand is repeated, the repetition is an indication that the implementation is at a lower level."

Weitere Softwaremetriken

CV	Number of Class Variables	
OV	Number of Object Variables	
NOA	Number of Attributes	CV + OV
WAC	Weighted Attributes Count (per class)	$\sum_{i=1}^{\lvert A\rvert} v_A(a_i)$
WMC	Weighted Methods Count (per class)	$\sum_{i=1}^{\lvert M\rvert} v_M(m_i)$

DIT	Depth of Inheritance
NOC	Number of Children
NORM	Number of Redefined Methods
Fan-in	Anzahl Klassen, die direkt auf die Klasse zugreift.
Fan-out	Anzahl Klassen, auf die die Klasse zugreift.

Maintainability Index

$$MI = 171 - 5{,}2 * \ln(HV) - 0{,}23 * CC - 16{,}2 * \ln(LOC) + 50 * \sin(\sqrt{2{,}4 * COM})$$

- HV = Halstead-Volumen
- CC = Zyklomatische Komplexität
- LOC = Durchschnittliche Anzahl Codezeilen pro Modul
- COM = Prozentualer Kommentaranteil pro Modul

- (Oman & Hagemeister, 1992), (Coleman et. al., 1994), (Heitlager et. al., 2007)

Was macht Funktion f()?

```
public class Main {
   static void f(List<?> list, int n) {
      if (n == 0) {
         System.out.println(list);
         return;
      }
      list = new ArrayList<>(list);
      for (int i = 0; true; i++) {
         f(list, n - 1);
         if (i == n) break;
         Collections.swap(list, list.size() - n - 1, list.size() - n + i);
      }
   }

   public static void main(String[] args) {
      List<?> list = IntStream.range(1, 5).boxed().collect(Collectors.toList());
      f(list, list.size() - 1);
   }
}
```

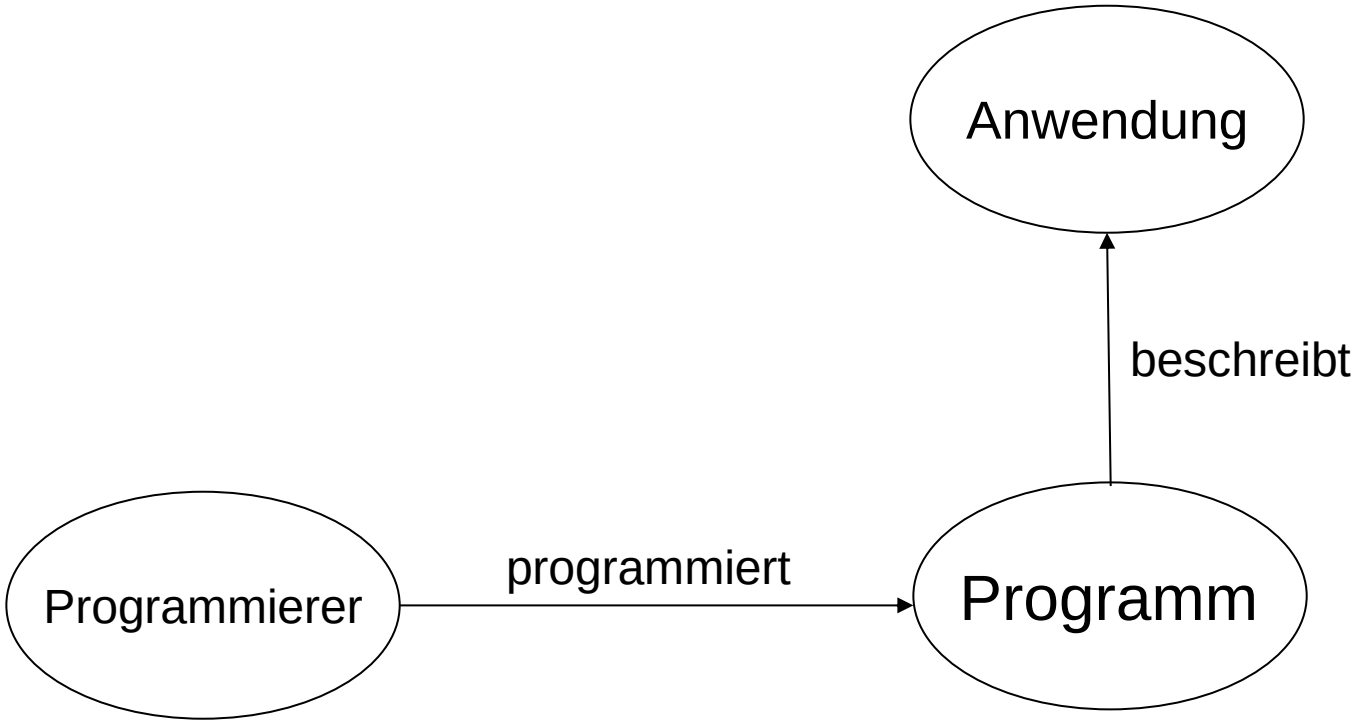
Ist das komplex?

```
public class CalculatorFrame extends JFrame {
   class CalcActionHandler implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         …
      }
   }
   …
   public CalculatorFrame() {
      calculator = new Calculator();
      setSize(300,80);
      JMenuBar menubar = new JMenuBar();
      …
      menu.add(mi = new JMenuItem("Berechnen"));
      mi.addActionListener(new CalcActionHandler());
      …
      panel.add(new JLabel("Funktion:"));
      panel.add(cboFunction = new JComboBox());
      cboFunction.addItem(new Addition());
      cboFunction.addItem(new Multiplication());
      panel.add(cmd = new JButton("Berechnen"));
      cmd.addActionListener(new CalcActionHandler());
      …
   }
   …
}
```

Komplexität von Programmen

- Derartige Softwaremetriken bestimmen die Komplexität, die Programme zu bewältigen haben.
- Lässt sich anhand dessen die Komplexität des Programmierens ableiten?

Charakteristik des Programmierens



Kritiken

- (Cant, et al., 1995):

> "…they begin with certain characteristics of the software and attempt to determine what effect they might have on the difficulty of the various programmer tasks. A more useful approach would be first to analyse the processes involved in programmer tasks, as well as the parameters which govern the effort involved in those processes."

- (Curtis, et al., 1979):

> "Yet, assessing the psychological complexity of software appears to require more than a simple count of operators, operands, and basic control paths. If the ability of complexity metrics to predict programmer performance is to be improved, then metrics must also incorporate measures of phenomena related by psychological principles to the memory, information processing, and problem solving capacities of programmers."

Kritiken

- (Shepperd, 1988) zeigt umfassend die Schwächen der zyklomatischen Komplexität auf. Er bezweifelt ihre Relevanz, insb. da sie in Studien stark mit LOC korrelierte und teilweise sogar schlechter abschnitt.

- In (Evangelist, 1983) wird festgestellt, dass von Kernighan und Plaugers Regeln guten Programmierstils (Kernighan & Plauger, 1978) nur 2 von 26 eindeutig zu einer geringeren zyklomatischen Komplexität führen.

Komplexität des Programmierens

- Anforderungen können möglichst direkt programmiert werden.
- Die Umsetzung ist redundanzfrei.
- Der Programmiererfolg wird durch die Entwicklungsumgebung möglichst geführt (z.B. Autovervollständigung durch IDE)
- Fehler werden von der Entwicklungsumgebung früh, umfassend und aussagekräftig kommuniziert (z.B. starke, statische Typsicherheit der Programmiersprache).
- Änderungen sind effizient umsetzbar.

Beispiele für Komplexität beim Programmieren

- …?

Wozu dann Softwaremetriken

- Softwaremetriken können automatisiert gemessen werden.
- Hohe Komplexität in Programmen können zu Reviews erfordern.
- Mögliche Reaktionen:
  - Rechtfertigung der Programmkomplexität basierend auf der Komplexität der Anforderungen
  - Refactoring bei Überkomplexität der Lösung
  - Anpassung der Anforderungen

Literatur

- Cant, S. N., Jeffrey, D. R. & Henderson-Sellers, B., 1995. A conceptual model of cognitive complexity of elements of the programming process. *Information and Software Technology*, 37(7), pp. 351-362.
- Coleman, D. M., et al., 1994. Using metrics to evaluate software system maintainability. *IEEE Computer*, 27(8), pp. 44-49
- Curtis, B. et al., 1979. Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics. *IEEE Transactions on Software Engineering*, pp. 96-104.
- Evangelist, W. M., 1983. Software complexity metric sensitivity to program structuring rules. *Journal of Systems and Software*, 3(3), p. 231–243.
- Halstead, M. H., 1977. *Elements of Software Science.* New York: Elsevier Science.

Literatur

- Heitlager, I., Kuipers T. & Visser, J., 2007. A Practical Model for Measuring Maintainability. *6th International Conference on the Quality of Information and Communications Technology*, pp. 30-39: https://www.softwareimprovementgroup.com/wp-content/uploads/2019/10/APracticalModelForMeasuringMaintainability.pdf
- McCabe, T. J., 1976. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2(4), pp. 308-320.
- Oman, P. & Hagemeister, J., 1992. Metrics for assessing a software system's maintainability. *Proceedings of Conference on Software Maintenance*, pp. 337-344
- Shepperd, M., 1988. A critique of cyclomatic complexity as a software metric. *Software Engineering Journal*, pp. 30-36.
- Sommerville, I., 2001. *Software Engineering, 6. Auflage.* München: Pearson.