Vorlesung (WS 2013/14)

Softwarekonstruktion

Prof. Dr. Jan Jürjens

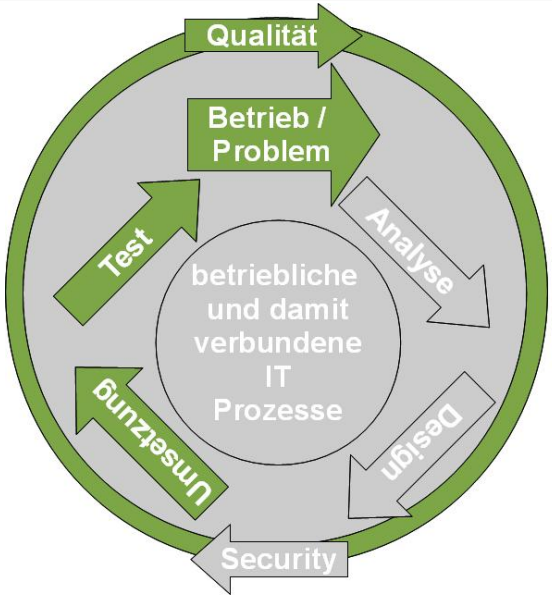
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

Teil 3.3: Softwaremetriken

v. 20.12.2013

Einordnung
Softwaremetriken

- Modellgetriebene SW-Entwicklung
- Qualitätsmanagement
- Softwareverifikation
  - Grundlagen Softwareverifikation
  - White-Box-Test
  - Softwaremetriken



[inkl. Beiträge von Prof. Martin Glinz, Universität Zürich und Prof. Ian Sommerville, Univ. St. Andrews]

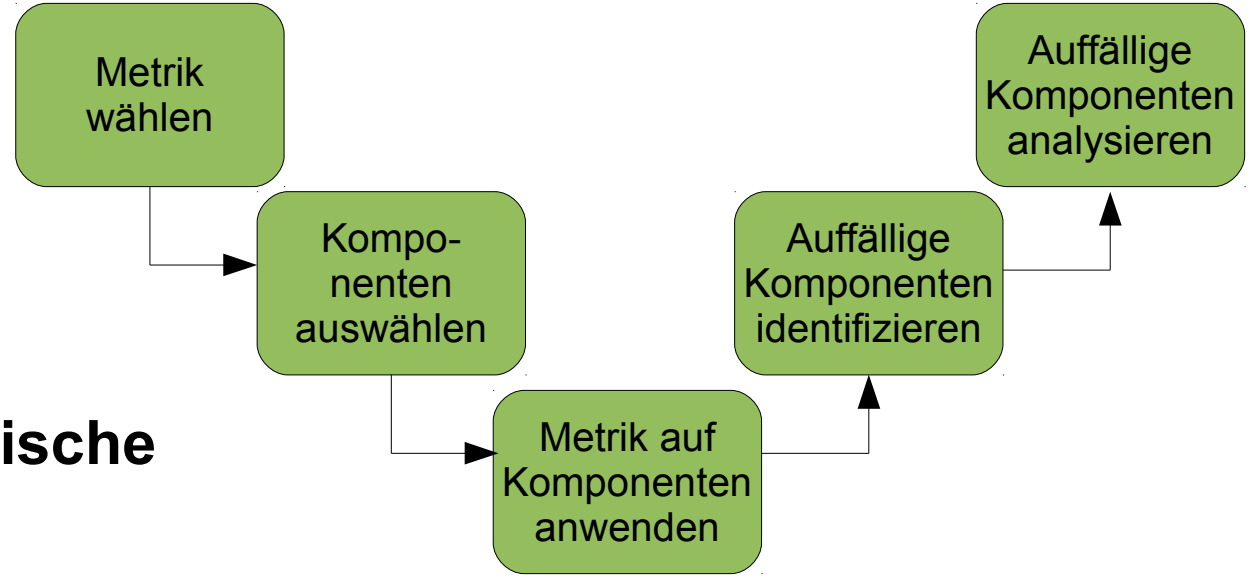
Literatur:

• Andreas Spillner, Tilo Linz: Basiswissen Softwaretest, Aus- und Weiterbildung zum Certified Tester Foundation Level nach ISTQB-Standard.

• Eike Riedemann: Testmethoden für sequentielle und nebenläufige Software-Systeme.

Softwareverifikation vs. Softwaremetriken

- Volllständiges Austesten i.A. unmöglich => **Prioritäten** setzen
- Schwerpunkt auf besonders **fehleranfällige** Teile der Software legen !
- Wie diese Teile effizient identifizieren ?
- Fehleranfälligkeit korreliert mit **Komplexität** von Softwarekomponente.
- Komplexität der Softwarekomponenten mit **Metriken** ermitteln.
- Metrikwerte für verschiedene Komponenten und mit historischen Daten **vergleichen**.
- Anomale Werte können auf Qualitätsprobleme hinweisen
  => intensiver testen.
- Hier ein Beispiel: **zyklomatische Komplexität**.



Zyklomatische Zahl eines zyklischen Graphen

LEHRSTUHL 14
SOFTWARE ENGINEERING

- **Zyklischer Graph:** Auffassung als Vektorraum.
  [via Menge seiner Eulerschen Teilgraphen]

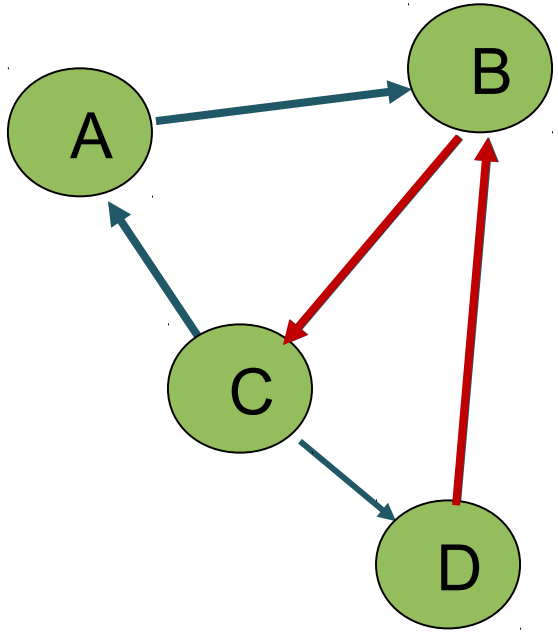
  → **Menge unabhängiger Pfade** (erzeugen in Linearkombination alle Pfade durch Graphen).

- **Zyklomatische Zahl:** Anzahl linear unabhängigen Pfade eines (zyklischen) Graphen.
- Sei **G = (V, E)** mit
  - e = |E(G)|, Anzahl Kanten (edges).
  - v = |V(G)|, Anzahl Knoten (vertices).
- Dann gilt **cn(G) = e - v + 1 (cyclomatic number).**

Zyklomatische Zahl: Alternative Definition

- **Alternative (äquivalente) Definition:** Anzahl zu entfernender Kanten stark zusammenhängenden Graphen, um **Spannbaum** zu erhalten.
- **Beispiel:**
  cn(G) = ?

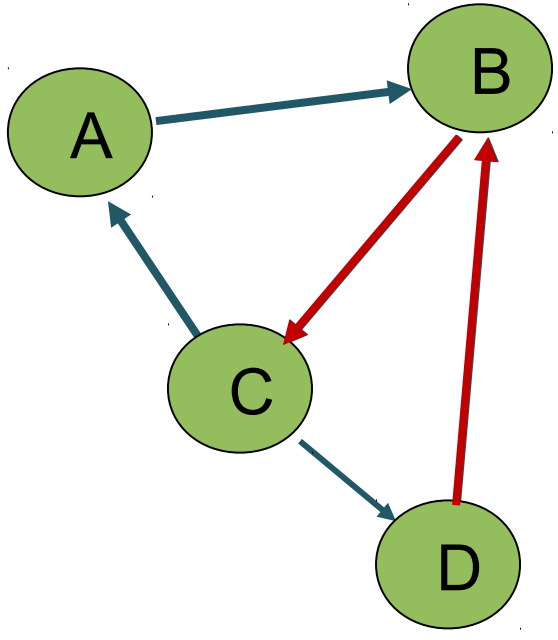
  (NB: **Entfernung beider Kanten** B-C und D-B ergibt Spannbaum des Graphen !)



Zyklomatische Zahl: Alternative Definition

- **Alternative (äquivalente) Definition:** Anzahl zu entfernender Kanten stark zusammenhängenden Graphen, um **Spannbaum** zu erhalten

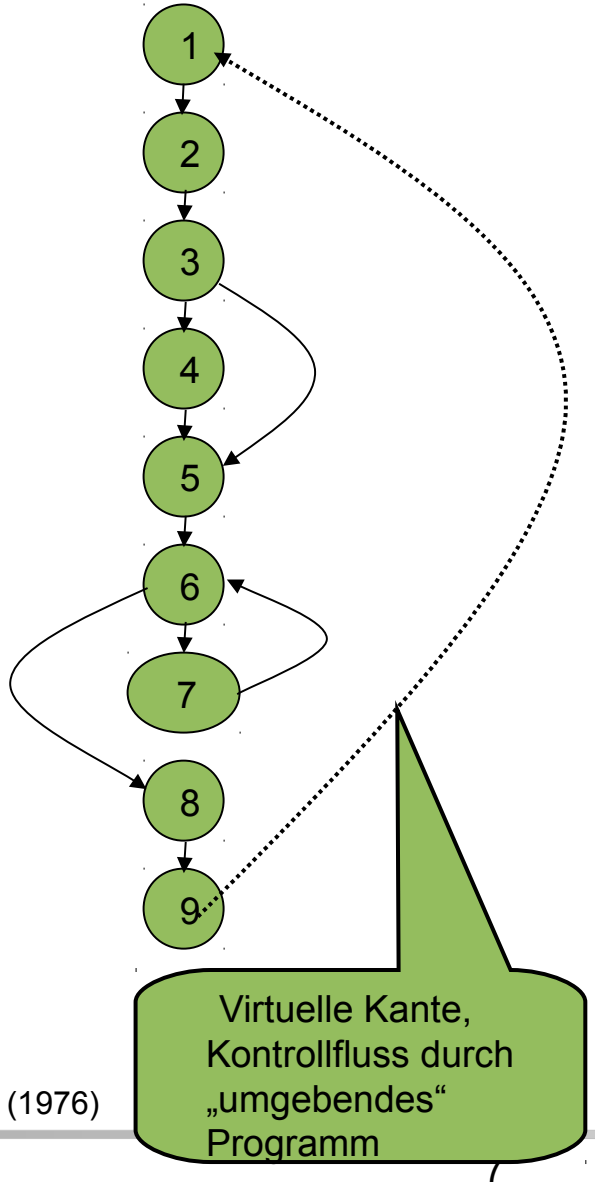
- **Beispiel:**
  cn(G) = 5 - 4 + 1 = 2



(NB: **Entfernung beider Kanten** B-C und D-B ergibt einen Spannbaum des Graphen !)

Zyklomatische Komplexität eines Programmes

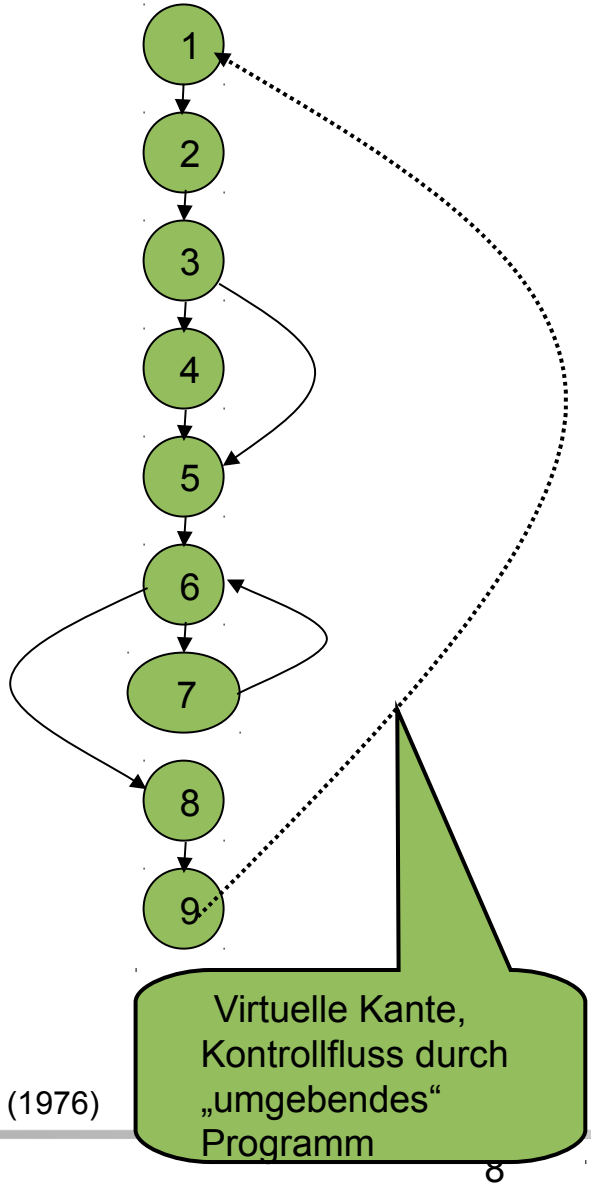
- Für **Definition dieser Zahl** als Metrik für Programme, Kontrollflussgraphen **„virtuelle“ Kante vom Endknoten zurück zum Anfangsknoten** hinzufügen, um zyklischen Graphen zu erhalten. (**Annahme:** Programm hat nur einen Endpunkt.)
- $\nu(G) = e - v + 2$: **Zyklomatische Komplexität** („McCabe-Metrik“) eines Kontrollflussgraphen G.
- **Beispiel:** $\nu(G) = ?$



T.J. McCabe: A Complexity Measure, IEEE Transactions on Software Engineering Vol. 2, No. 4, p. 308 (1976)

Zyklomatische Komplexität eines Programmes

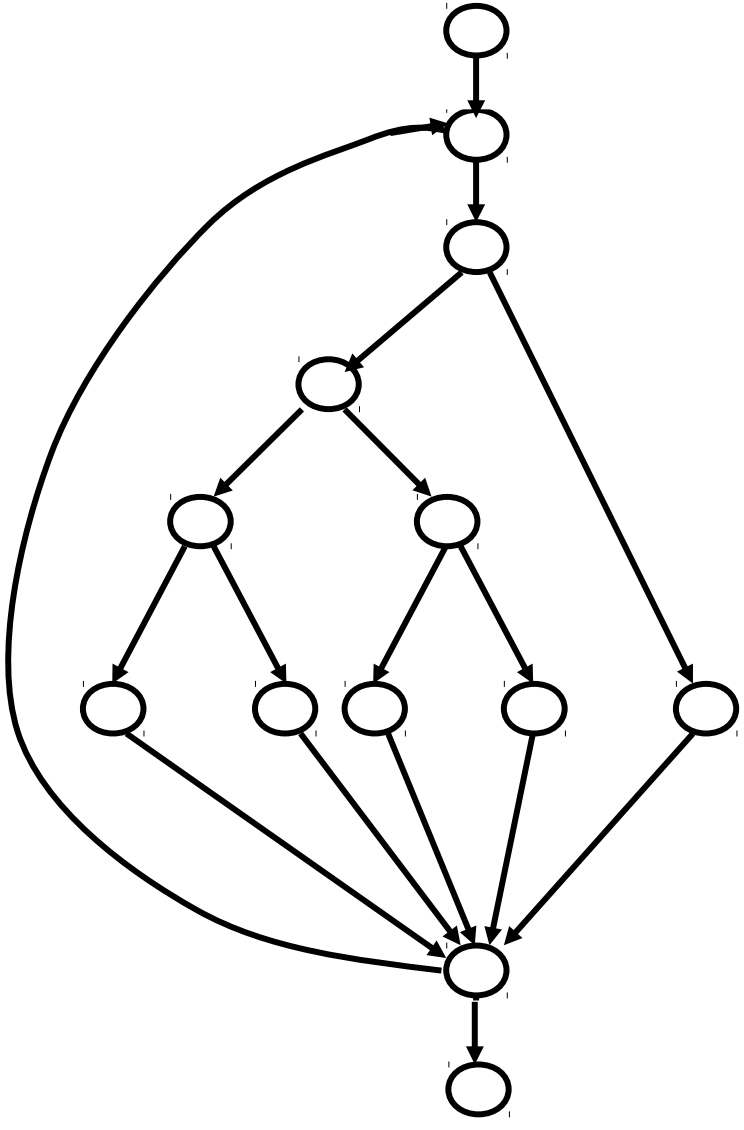
- Für **Definition dieser Zahl** als Metrik für Programme, Kontrollflussgraphen **„virtuelle“ Kante vom Endknoten zurück zum Anfangsknoten** hinzufügen, um zyklischen Graphen zu erhalten. (**Annahme:** Programm hat nur einen Endpunkt.)
- $\nu(G) = e - v + 2$: **Zyklomatische Komplexität** („McCabe-Metrik“) eines Kontrollflussgraphen G.
- **Beispiel:** $\nu(G) = 10 - 9 + 2 = 3$.



T.J. McCabe: A Complexity Measure, IEEE Transactions on Software Engineering Vol. 2, No. 4, p. 308 (1976)

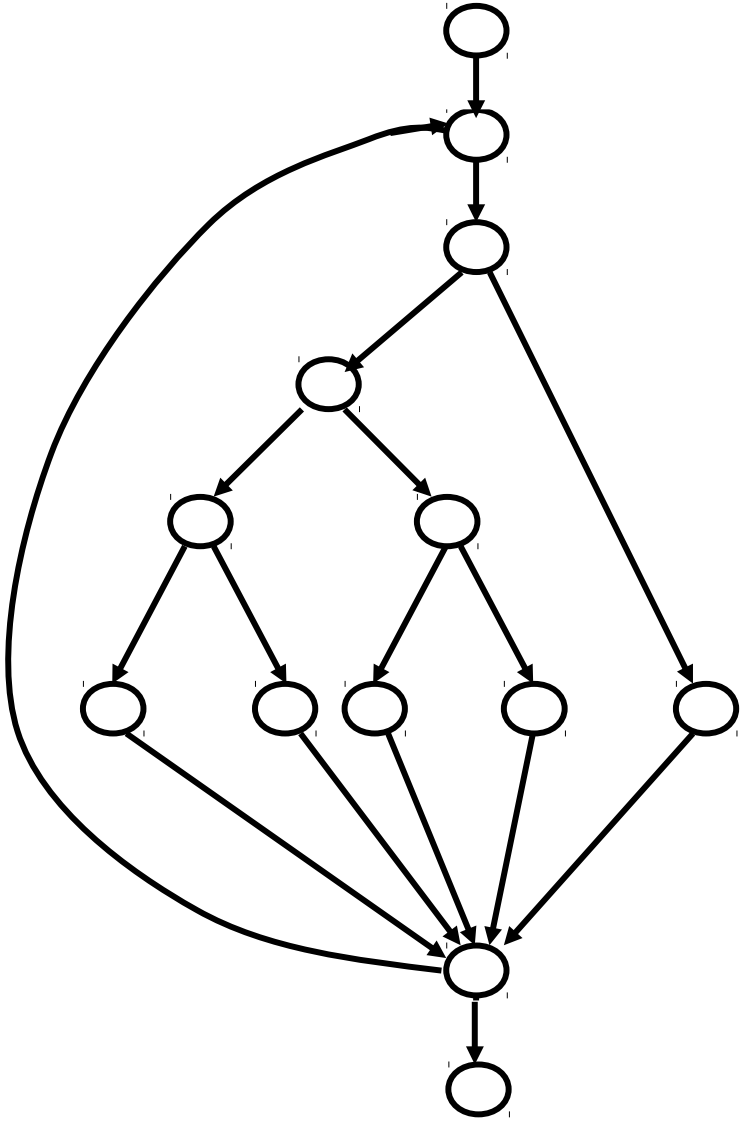
Beispiel: Zyklomatische Komplexität

- **Lösung:**
  v = ? , e = ?
  $\nu(G) = e - v + 2 = ?$
- **Bemerkung:** Berechnung von $\nu(G)$ in Programmiersprachen mit geschlossenen Ablaufkonstrukten:
  - Zähle alle Verzweigungen und Schleifen (**if**, **while**, **for**, etc.).
  - Addiere für jede Auswahl-Anweisung (**switch**, **CASE**) Zahl der Fälle - 1.
  - Addiere 1.
- **Hier:** ? + 1



Beispiel: Zyklomatische Komplexität

- **Lösung:**
  v = 13, e = 17
  $\nu(G) = e - v + 2 = 17 - 13 + 2 = 6$
- **Bemerkung:** Berechnung von $\nu(G)$ in Programmiersprachen mit geschlossenen Ablaufkonstrukten:
  - Zähle alle Verzweigungen und Schleifen (**if**, **while**, **for**, etc.).
  - Addiere für jede Auswahl-Anweisung (**switch**, **CASE**) die Zahl der Fälle - 1.
  - Addiere 1.
- **Hier:** 5 + 1.



Mehrere Endpunkte

Bislang Annahme: Programm hat nur ein Endpunkt.

Programm mit mehreren Endpunkten: Für jeden Endpunkt „virtuelle“ Kante zum Startpunkt einfügen. → Zyklischen Graph erhalten.

Zyklomatische Komplexität des Kontrollflussgraphen G eines Programms **mit mehreren Endpunkten**:

$\nu(G) = e - n + p + 1$

- e Zahl der Kanten
- n Zahl der Knoten
- p Zahl der Endpunkte des Programms

[NB: Annahme weiterhin: Nur ein Startpunkt.]

Bewertung der zyklomatischen Komplexität (1)

- Zyklomatische Komplexität **höher als 10** nach McCabe **nicht tolerabel.**
  - Überarbeitung des Programmteils !
  - Messwert 6 im Beispiel liegt im Bereich, den McCabe für akzeptabel hält.
  - Zum Teil Messwerte bis 15 ausnahmsweise akzeptiert (bei dokumentierter Begründung).
- **Für Wartbarkeit: Verständlichkeit** eines Programmstücks von entscheidender Bedeutung.
  - Je **höher ermittelte zyklomatische Komplexität**, desto schwieriger Nachvollziehen des Ablaufs des Programmstücks und desto **schlechter Verständlichkeit.**

Bewertung der zyklomatischen Komplexität (2)

Auskunft über Testaufwand:

- Zyklomatische Komplexität = **Anzahl unabhängiger Pfade** im Programmstück.
- Zyklomatische Komplexität – 1 = **Anzahl Entscheidungen im Kontrollflussgraph** eines strukturierten Programms.
- **100%-ige Ausführung aller Anweisungen** und Verzweigungsmöglichkeiten eines Programms verlangt.
  → Dafür könnten alle unabhängigen Pfade durch Kontrollflussgraphen einmal »durchlaufen« werden.
- Zyklomatische Komplexität: **Obere Grenze für Anzahl benötigter Testfälle** zur Erreichung dieses Kriteriums.

Bewertung der zyklomatischen Komplexität (3)

LEHRSTUHL 14
SOFTWARE ENGINEERING

- Problem der **Validität:**
  - Spaghetti-Programm und wohlstrukturiertes Programm gleichen Problems können gleiche zyklomatische Komplexität haben.
  - Sind diese Programme auch intuitiv gleich komplex ?
- Eignet sich Maß als **Indikator für Fehleranfälligkeit ?**

  → Nicht besser als NCSS (Kafura und Canning, 1985).
- Skala: **Verhältnisskala**, aber **nicht additiv:** Aneinanderreihung zweier Programmstücke mit Komplexitäten ν1 und ν2 hat Komplexität ν1+ν2-1: **Kontraintuitive Eigenschaft.**

Die objektorientierte Metriken-Suite „CK“ (1)

Softwaremetrik	Beschreibung
Gewichtete Methoden pro Klasse (Weighted Method Comple-xity, WMC)	**Anzahl Methoden in jeder Klasse**, gewichtet durch Komplexität: WMC = $\Sigma C(i)$ mit C(i) = Komplexität von Methode i **Einfache Methode** kann **Komplexität 1** haben; größere und komplexere sehr viel höheren Werte. **Komplexe Objekte:** Meistens schwerer zu verstehen.
Tiefe des Vererbungs-baums (Depth of Inheritance Tree, DIT)	**Maximale Tiefe der Generalisierungshierarchie**. Repräsentiert Anzahl Ebenen im Vererbungsbaum, in dem Unterklassen Attribute und Operationen (Methoden) von Superklasse erben. Je tiefer der Baum, desto komplexer ist Design: Man muss viele Klassen verstehen, um unterstes Blatt des Vererbungsbaums nachvollziehen zu können.
Zahl der Kinder (Number of Children, NOC)	**Anzahl direkter Unterklassen.** Maß für unmittelbare Unterklassen einer Klasse. Misst **Breite einer Klassenhierarchie**, während DIT Tiefe repräsentiert. **Hoher NOC-Wert** kann **hohe Wiederverwendung** anzeigen.Mehr Anstrengung in Validierung der Basisklassen fließen, da Zahl der Unterklassen, die von ihnen abhängen, groß ist.

Die objektorientierte Metriken-Suite „CK“ (2)

Softwaremetrik	Beschreibung
Kopplung von Klassen (CBO)	Klassen C und D sind gekoppelt, wenn Methoden von C Methoden oder Variablen von D benutzen. **CBO: Maß** dafür, **wie viele Kopplungen existieren.** Hoher CBO-Wert zeigt, dass Klassen im hohen Maße voneinander abhängig sind. Damit ist es wahrscheinlicher, dass **Änderung einer Klasse viele Klassen im Programm betrifft.**
Reaktion einer Klasse (RFC)	**Maß für Anzahl Methoden**, die potentiell als **Antwort auf Nachricht** an umgebene Klasse ausgeführt werden können. **RFC: Mit Komplexität verbunden.** Je höher der RFC-Wert, desto komplexer und wahrscheinlich fehleranfälliger ist eine Klasse.
Mangel an Zusammenhalt in Methoden (Lack of Cohesion of Methods, LCOM)	**Anzahl durch Methoden** einer Klasse gemeinsam **benutzten Instanzvariablen.** Differenz der Anzahl von Methodenpaaren mit geteilten Attributen und Anzahl ohne diese. Diese Metrik existiert in verschiedenen Variationen. Es ist nicht klar, ob dieses Maß irgendeine zusätzliche Information zu anderen Metriken liefert.

Diskussionsfrage

Komplexitätsmaße: Unter anderem Indikator für Fehleranfälligkeit und Pflegbarkeit. Welches **Problem** ergibt sich, wenn Programmierer am **Komplexitätsmaß gemessen** wird, um beide Eigenschaften zu steuern ?

Diskussionsfrage

Komplexitätsmaße: Unter anderem Indikator für Fehleranfälligkeit und Pflegbarkeit. Welches **Problem** ergibt sich, wenn Programmierer am **Komplexitätsmaß gemessen** wird, um beide Eigenschaften zu steuern ?

Antwort:

1) Komplexität von Code ergibt sich aus Komplexität des zu lösenden Problems und somit nur teilweise durch Programmierer steuerbar. Wenn man **mehrere Lösungen** für das **gleiche (oder ähnliche) Problem** durch Metriken vergleicht, kann man **Qualität der Programmierung vergleichen.**

2) Wenn **Metrik bekannt** ist, kann Programmierer **Code optimieren,** ohne notwendigerweise Qualität zu verbessern.

Werkzeugunterstützung

Testwell CMTJava http://www.testwell.fi/cmtjdesc.html

- *Unterstützte Metriken:* Zeilenmetriken, **Halstead-Metrik, McCabe Zyklomatische Komplexität,** Wartungsaufwand.

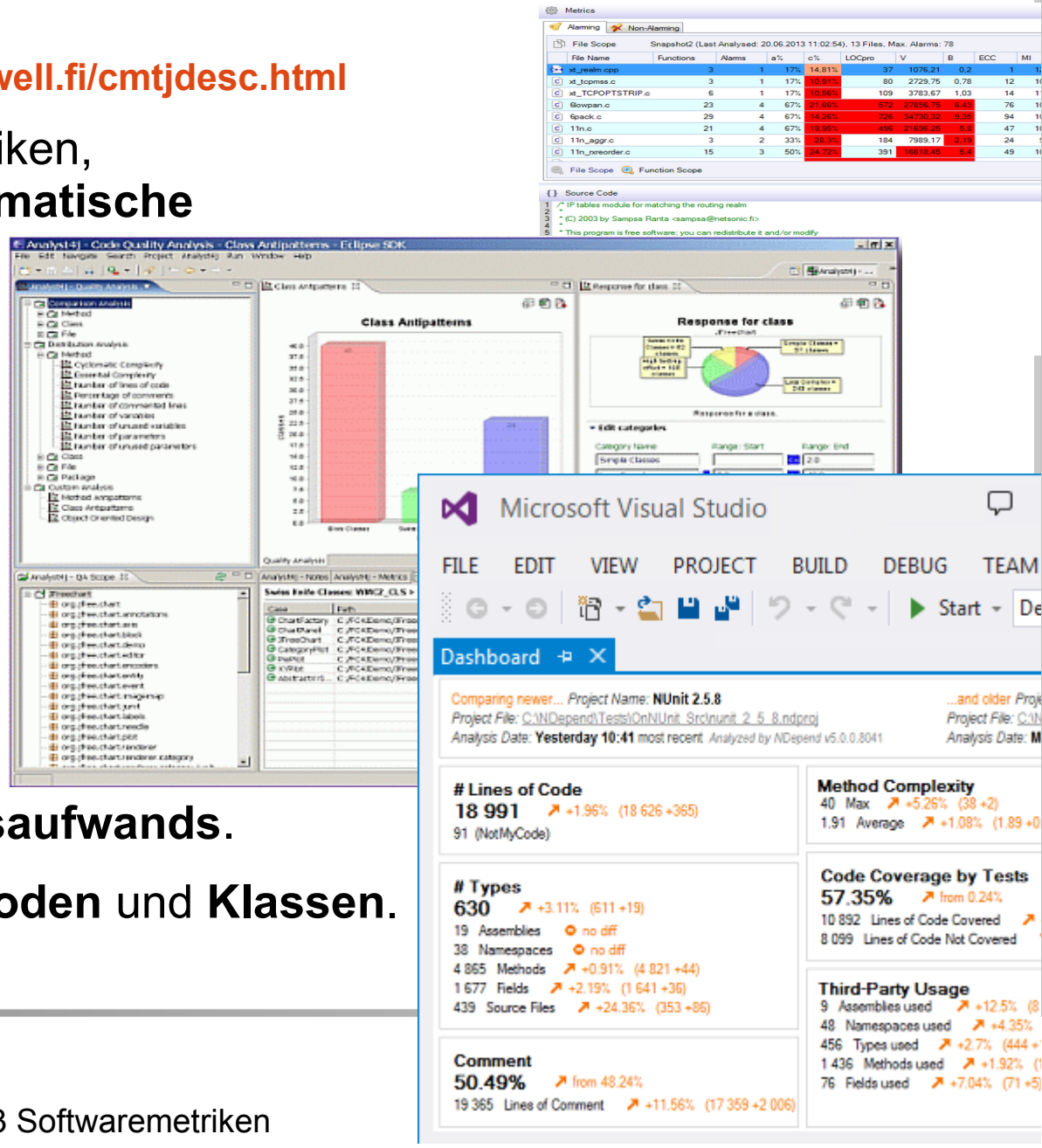
Eclipse Metrics Plugin

http://eclipse-metrics.sourceforge.net

- *Unterstützte Metriken:* **McCabe Zyklomatische Komplexität.**

Ndepend http://ndepend.com

- *Unterstützte Metriken:*
  - **Zyklomatische Komplexität**
  - Schätzung des **Entwicklungsaufwands**.
  - Erkennung von **großen Methoden** und **Klassen**.

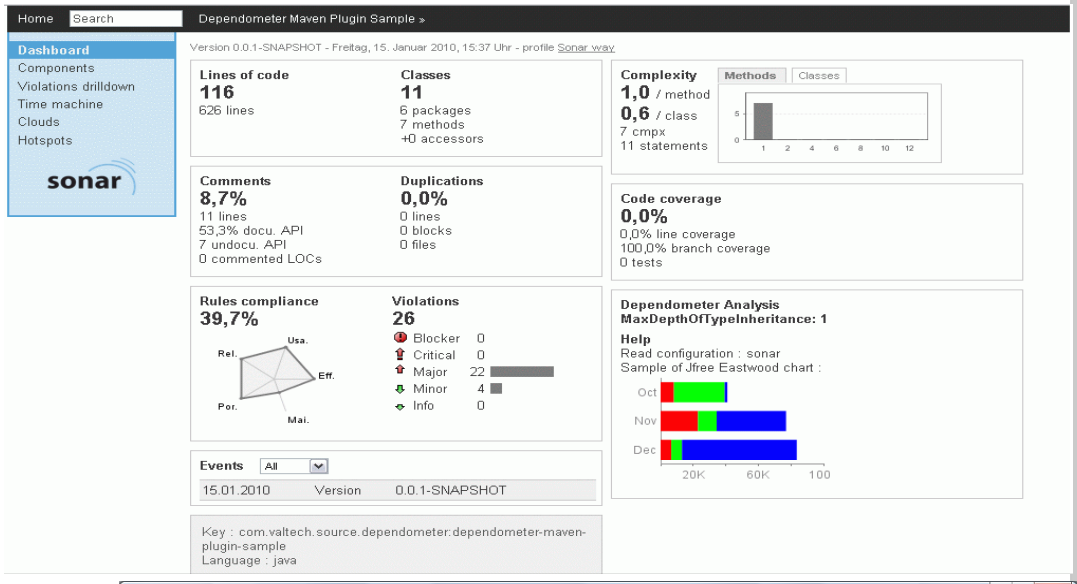


Werkzeugunterstützung

Sonar Source

http://sonarsource.com

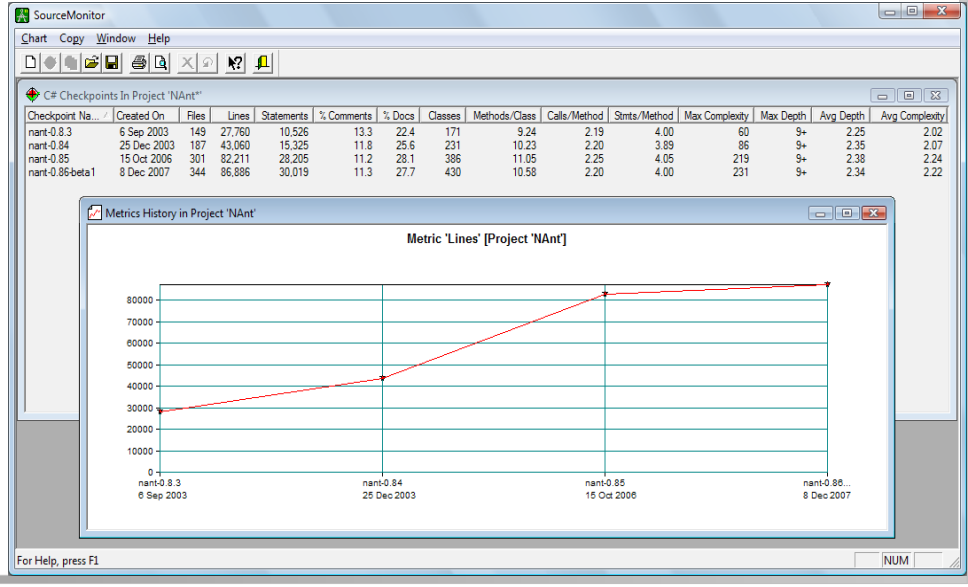
- *Unterstützte Metriken:* **McCabe Zyklomatische Komplexität.**



Source Monitor

http://www.campwoodsw.com/sourcemonitor.html

- *Unterstützte Metriken:* **Zyklomatische Komplexität.**



Anhang
(weitere Informationen)

Größenmaße

- Größe – Wie umfangreich ist Software?
- Skala: Verhältnisskala.
- Mögliche Maße: **NCSS, Anzahl Zeichen.**

NCSS (Non-commented source statements):

- Zählung der **Codezeilen** ohne Kommentar- und Leerzeilen.
- Genaue **Zählregeln** erforderlich.
- **Programmiersprachenabhängig.**
- **Leicht messbar.**

Weitere Komplexitätsmaße

Softwaremetrik	Beschreibung
Fan-in / Fan-out	**Fan-in** einer Funktion oder Methode X: Anzahl Funktionen oder Methoden, die X aufrufen. **Fan-out:** Anzahl Funktionen oder Methoden, die von X aufgerufen werden. **Hoher Fan-in:** X eng mit Rest des System verbunden. Änderungen an X können weitreichende Folgewirkungen haben. **Hoher Fan-out:** Gesamte Komplexität von X ist hoch, da Steuerungs-logik von X aufgerufene Komponenten koordinieren muss.
Länge der Bezeichner	Maß durchschnittlicher Länge von Bezeichnern (Namen von Variablen, Klassen, Methoden, etc.) im Programm. Je länger sie sind, desto aussagekräftiger sind sie (damit Programm verständlicher).
Tiefe der Verschachte-lung	Maß der Tiefe der Verschachtelung von if-Bedingungen im Programm. Tief verschachtelte if-Bedingungen: Schwer zu verstehen und potentiell fehleranfälliger.
Fog-Index	Maß durchschnittlicher Länge von Wörtern und Sätzen im Dokument. Je höher Fog-Index eines Dokuments ist, desto schwerer ist es zu verstehen.

Weitere Metriken für objektorientierte Programme

Kürzel	Bezeichnung	Erläuterung
NOV	Number of Variables	Anzahl der Instanzvariablen (member variables) einer Klasse
NOM	Number of Methods	Anzahl der Methoden (Operationen) einer Klasse
NORM	Number of Redefined Methods	Anzahl der in einer Klasse redefinierten Methoden