Metriken für den Objektorientierten Entwurf

Alexander Ostrovsky
ostrovsk@in.tum.de

Gliederung

- Grundlagen
- Objektorientierter Entwurf
- Metriken für den objektorientierten Entwurf
- Packagemetriken
- Goal-Qustion-Metric-Ansatz
- Automatisierung

GRUNDLAGEN

Wiederholung: Wozu Messen?

- Qualitätseigenschaften messbar von:
  - Software
  - Entwicklungsprozess
- Möglichkeit rechtzeitig in den Entwicklungsprozess einzugreifen
- Güte eines Produktes Messen
  - Z.B. Bei der Einführung eines neuen Produktes.

Was ist eine (Software-)Metrik

„Eine Softwarequalitätsmetrik ist eine Funktion, die eine Software-Einheit in einen Zahlenwert abbildet. Dieser berechnete Wert ist interpretierbar als der Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit.“
(IEEE 1061, 1992)

Qualitätseigenschaften?

- Qualitätseigenschaften: Fehlerfreiheit, Zuverlässigkeit, Effizienz, Benutzungsfreundlichkeit, Wartbarkeit, …
- Innere Qualität
  - Wartbarkeit, Wiederverwendbarkeit,…
- Äußere Qualität
  - Effizienz, Stabilität, Benutzerfreundlichkeit,...

Nutzen & Kritik von Metriken

- Unterstützen die Suche nach Problemen im Quellcode
- Softwareentwicklung vorhersehbarer
- Erzieherischer Effekt
- **ABER**: Schlechte Messwerte nur Indizien, keine handfesten Belege
- Viele Eigenschaften nicht direkt messbar.
- Möglichkeit der subjektiven Einwirkung

Beispiel

```
String wochentagsName(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)"
}
```

- Schlechte McCabe Komplexität von 9, aber eigentlich übersichtlich

Gütekriterien

- Objektivität
  - Heute oft gegeben, da viele Metriken sich mathematisch aufschreiben lassen
- Robustheit
  - Gleiche Eingabewerte -> Gleiche Ausgabewerte
- Vergleichbarkeit
- Ökonomie
- Korrelation
  - Aus den Messergebnissen kann ein Rückschluss gezogen werden.
- Verwertbarkeit
  - Die Ergebnisse sollen das Handeln beeinflussen.

OBJEKTORIENTIERTER ENTWURF (OOD)

Prozess des OOD

1. Identifikation der Klassen
   - Suche nach Schlüsselabstraktionen
2. Identifikation der Semantik
   - Bedeutung + Lebenszyklus
3. Identifikation der Klassenbeziehungen
   - Beziehungen + Vererbungshierarchie + Sichtbarkeiten
4. Festlegung der Funktionen und des Verhaltens

- Messung der Güte von 1 – 3 -> Dieser Vortrag

Ziele des OOD

- Verminderung des Wartungsaufwandes
  - Vereinfachung der Einführung neuer Funktionen
  - Vereinfachung der Änderung der Funktionen
- Dafür muss sein:
  - Geringe Kopplung
  - Hohe Kohäsion

METRIKEN FÜR OOD

Bedingte Eignung der prozeduralen Metriken

- Metriken wie LOC oder McCabe eignen sich nicht immer für OOD
  - Keine Bewertung der Güte des OOD
  - Keine Aussage über die Klassenbeziehungen
- Trotzdem lassen sich klassische Metriken für die Bewertung der einzelnen Funktionen verwenden

Typen von OOD-Metriken

- Komponentenmetriken
  - Bewertet Bestandteile wie Klassen, Module oder Pakete
    - Zählung der der Variablen und Untersuchung der Ausreiser
    - Vererbungsmetriken
- Strukturmetriken
  - Bewertet den Zusammenhang der Klassen untereinander
    - Wie oft greift eine Klasse auf die andere zu (Fan-Out)
    - Wie oft wird auf eine Klasse zugegriffen (Fan-In)

KONKRETE METRIKEN

Konkrete Metriken

- Chidamber-Kemerer-Metriken
  - Weighted Methods per Class (WMC)
  - Depth of Inheritance Tree (DIT)
  - Number of Children (NOC)
  - Coupling between object classes (CBO)
  - Lack of Cohesion in Methods (LCOM)
  - Response For a Class (RFC)
- Li & Henry – Metriken
  - “Erweiterte” Kopplung
  - SIZE-Metriken (z.B. Anzahl der Semikolons)
  - Li, W., & Henry, S. (1993). Object-oriented metrics that predict maintainability

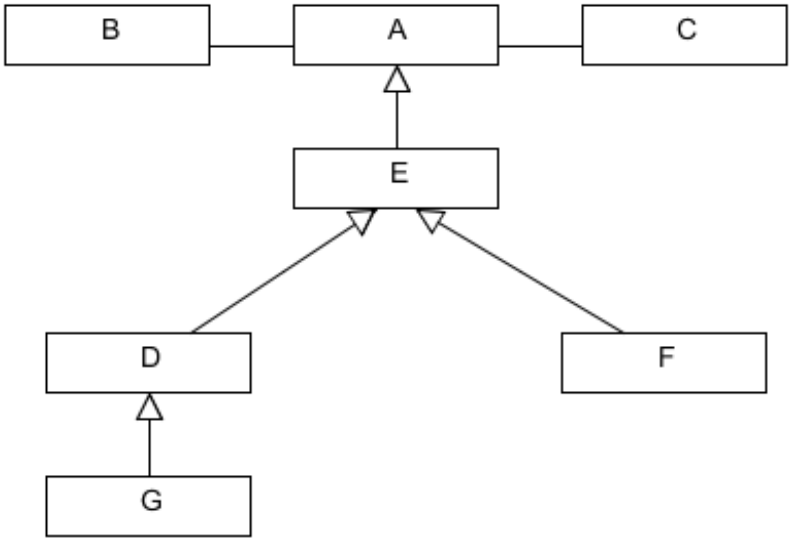
Weighted Methods per Class

- Summe der Komplexitäten der einzelnen Methoden
  - Z.B. mit McCabe
  - Komplexität 1 -> WMC die Anzahl der Methoden
- Hoher WMC-Wert->
  - Programmspezifisch
  - Schlechter wiederverwendbar
  - Höherer Wartungs- und Entwicklungsaufwand

Depth of Inheritance Tree

- $DIT(C)$ = (Größter) Abstand von der Wurzel zur Klasse C
- $DIT(C)$ groß -> viele Klassen können C beeinflussen
- Klassenhierarchie groß ->
  - Viele Vererbungen
  - Schlechtes Codeverständnis
  - Erhöhung der Komplexität des Objektentwurfes
  - Klassen unten im Vererbungsbaum schwer wiederverwendbar

Beispiel

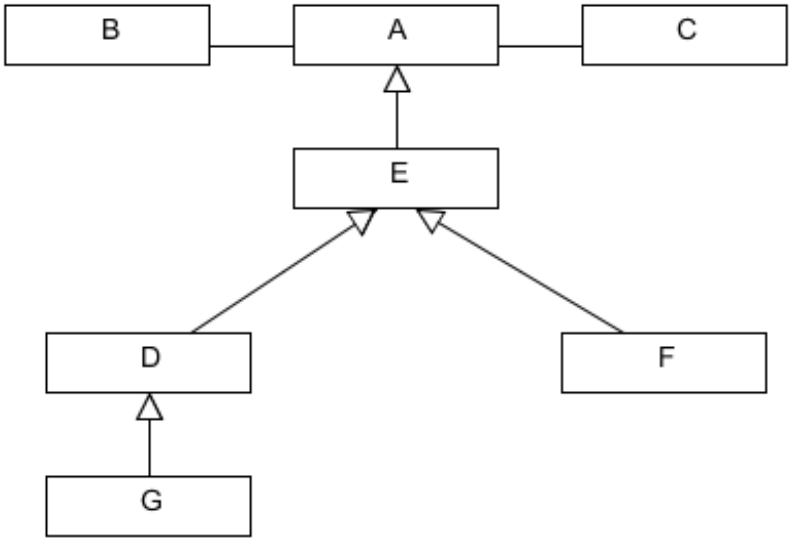


- $DIT(A) = 0$
- $DIT(G) = 3$
- In Java DIT meist ≥ 1, da Vererbung von Object

Number of Children

- Breite des Vererbungsbaumes
- Konkret: $NOC(C)$ = Anzahl der direkten Kinder von C
- NOC groß ->
  - Methoden oft wiederverwendet
  - Erhöhte Wahrscheinlichkeit des falschen Gebrauchs der Abstraktion
  - Höherer Testaufwand

Beispiel

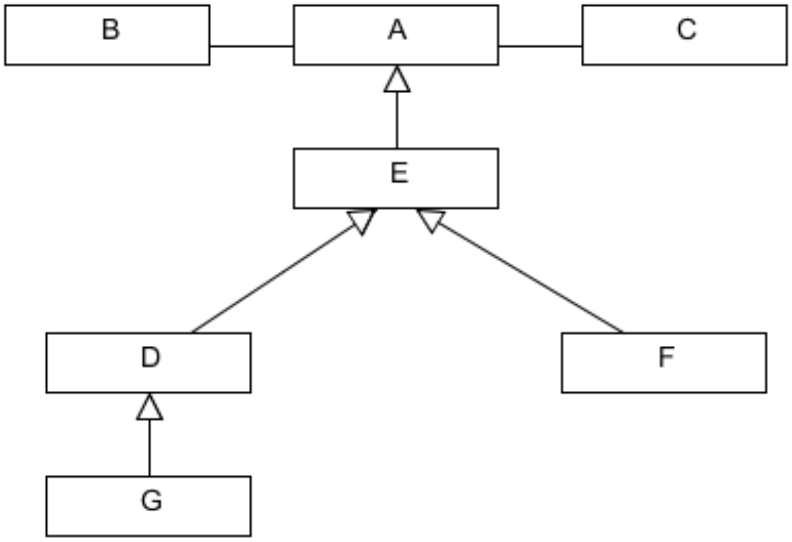


- $NOC(A) = 1$
- $NOC(E) = 2$

Coupling between object classes

- Anzahl der an Klasse C gekoppelten Klassen
  - Das sind die Klassen, die Methoden oder Instanzvariablen von C nutzen
- Exzessive Kopplung führt zu
  - Einschränkung der Wiederverwertbarkeit
  - Erhöhung des Testaufwandes
  - Einschränkung der Wartbarkeit

Beispiel



- $CBO(A) = 2$

Lack of Cohesion in Methods

- Anzahl der Paare von Methoden in einer Klasse ohne gemeinsame Attribute minus die Anzahl der Paare mit gemeinsamen Attributen jedoch ≥ 0
- Max (|{ohne gem. Attr.}|-|{mit gem. Attr.}|, 0)
- Hohe Kohäsion zeugt von
  - Guter Kapselung
  - Niedriger Komplexität

Beispiel

```
class AB {
    int a; int b;
    int a() {return a;}
    int b() {return b;}
    int bb() {return b * b;}
    int bbb() {return b * b * b;}
}
```

- Mit gemeinsamen Attribut:
  - (b,bb); (b,bbb); (bb,bbb)
- Ohne gemeinsame Attribute
  - (a,b); (a,bb); (a,bbb)
- Also: LCOM(AB)=0
  - Trotzdem lässt sich AB in 2 Klassen aufteilen

Response for a Class

- $RFC(C)$ ist die Anzahl der Methoden in C + die Anzahl der direkt aufgerufenen Methoden in anderen Klassen.
- RFC hoch heißt oftmals:
  - Zu viel Kopplung
  - Mehr Testauswand
  - Höherer Debugging- und Wartungsaufwand

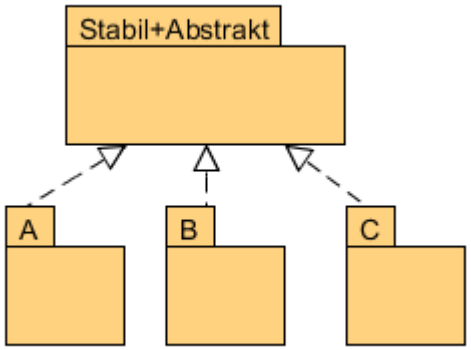
Beispiel

```
class Stack {
    //Default Constructor
    private List s = new ArrayList();
    void push(Object element) {s.add(element)  ;}
    Object pop() {return s.remove(s.size()  1);}
}
```

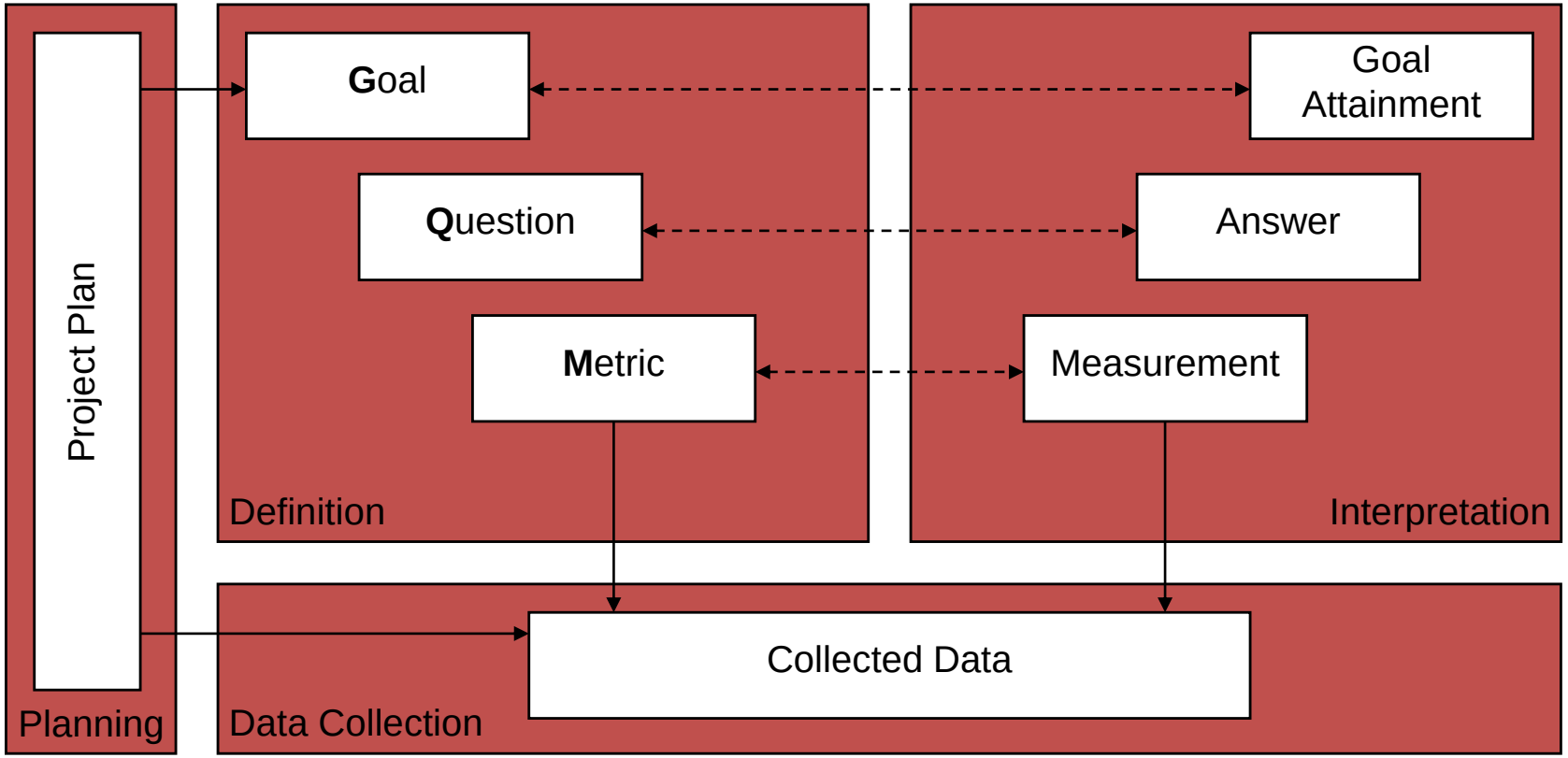
- $RFC(Stack) = 7$

Packagemetriken

- Kohäsion
- Afferent / Efferent Couplings
- Die Pakete sollen von stabileren und abstrakteren Pakten abhängen
  - Leichter zu ändern und zu erweitern



Goal-Question-Metric-Ansatz



[Wallmüller, Ernest: „SW-Qualitätsmanagement in der Praxis“. Hanser, 2001]

Automatisierung

- Siehe Eclipse

Fazit

- Vielfältige Einsatzmöglichkeiten
- Testen von Qualitätseigenschaften wie Wartbarkeit, Transparenz, Testbarkeit oder hier auch Grad der Objektorientierung
- Automatisiert (also günstig) erhebbar
- Jedoch oft zu wenig eingesetzt

Literatur I

- (kein Datum). Abgerufen am 2. April 2010 von Wikipedia: http://de.wikipedia.org/wiki/McCabe-Metrik
- Chiamber Shyam R., K. C. (6. 6 1994). A Metrics Suite for Object Oriented Design. *IEEE Transaction on Software Engineering Vol. 20*, S. 476-493.
- Chidamber Shyam R., K. C. (1991). Towards a Metrics Suite For Object Oriented Design. Cambridge, Massachusetts, USA.
- Elmer, F.-J. (2005). Abgerufen am 5. April 2010 von Universität Basel: informatik.unibas.ch/lehre/ws05/cs203/softeng13.pdf
- Franzen, B. (kein Datum). *SWT: Qualität*. Abgerufen am 1. April 2010 von Fachhochschule Gießen-Friedberg: http://homepages.fh-giessen.de/~hg7132/swt/skr9910/qualitat.html
- Gorman, J. (kein Datum). Abgerufen am 1. April 2010 von parlez | uml: www.parlezuml.com/metrics/OO%20Design%20Principles%20&%20Metrics.pdf
- Hoffmann, D. W. (2008). *Software-Qualität.* Berlin, Heidelberg: Springer Verlag.

Literatur II

- Li, W., & Henry, S. (1993). Object-oriented metrics that predict maintainability. *J. Syst. Softw.*
- Matthes, F. (kein Datum). *sebis: EIST.* Abgerufen am 1. April 2010 von Tu München: http://wwwmatthes.in.tum.de/wikis/sebis/eist
- Mills, E. E. (kein Datum). Abgerufen am 1. April 2010 von Carnegie Mellon University: http://www.sei.cmu.edu/reports/88cm012.pdf
- Taentzer, G. (29. April 2009). *Lehrveranstalltung Softwarequalität.* Abgerufen am 2. April 2010 von Philipps-Universität Marburg: http://www.mathematik.uni-marburg.de/~swt/ss2009/sq/files/Folien0429.pdf
- Wallmüller, E. (2001). *Software-Qualitäts-Managment in der Praxis.* München: Carl Hanser Verlag.