University of Basel

Software Engineering - Prinzipien

M. Lüthi, Universität Basel, 10. August 2022

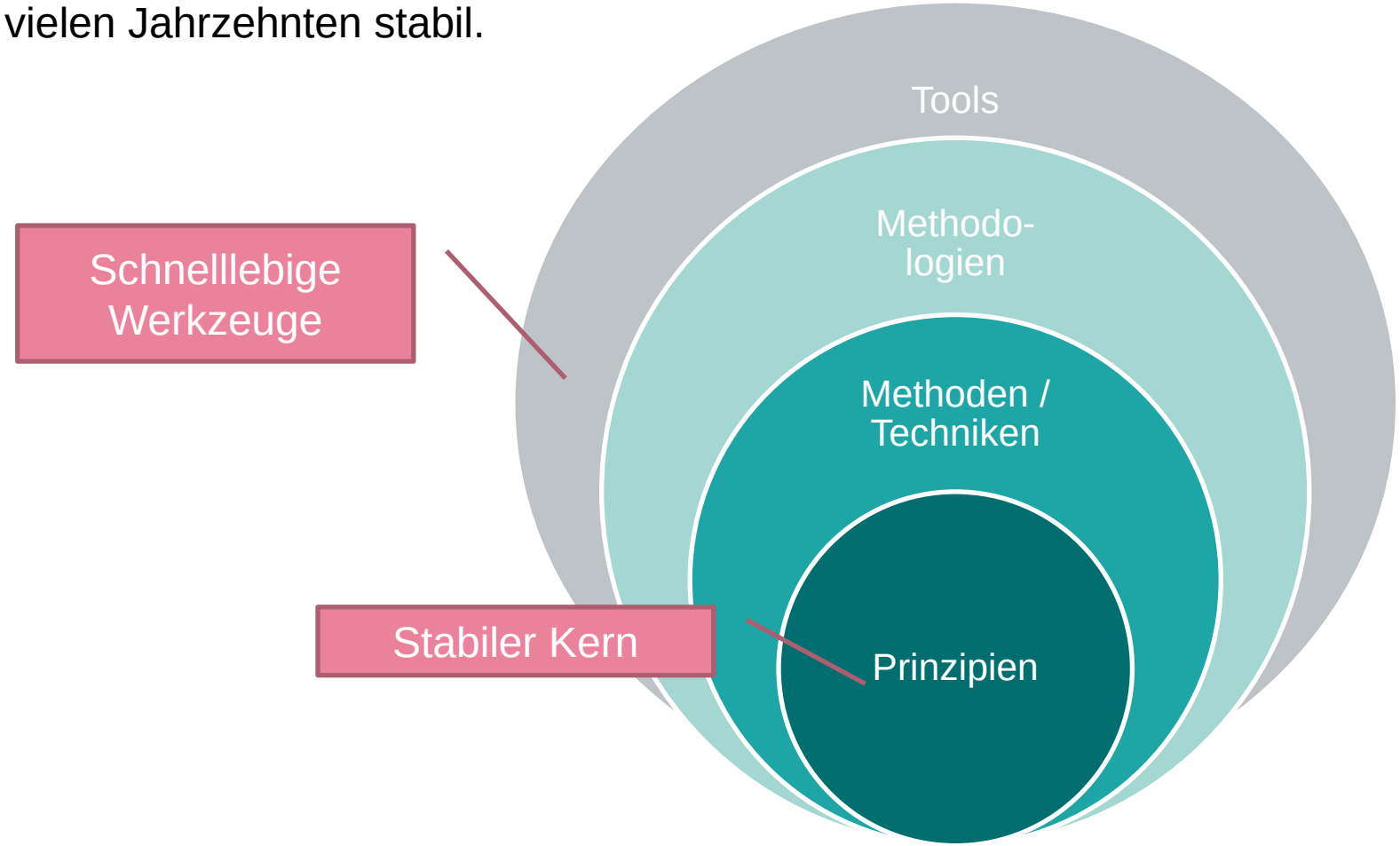
Agenda 10. August 2022

1	Prinzipien des Software Engineerings (30’)
2	Diskussion / Reflexion (30’)
3	Module und Modulbeziehungen (30’)
4	Einführung Buildsysteme (30’)
5	Praktische Übung 3: Gradle und Code-Reading (80’)

Prinzipien des Software Engineerings

Prinzipien des Software Engineerings

Technologien und Werkzeuge der Informatik entwickeln sich rasant.
Grundlegende Prinzipien sind seit vielen Jahrzehnten stabil.



Eigenschaft guter Prinzipien

Gute Prinzipien sollten abstrakt und falsifizierbar sein.

Beispiele

- Benutze JUnit um Tests zu schreiben 
- Schreibe qualitativ gute Software 
- Schreibe Tests immer zuerst 

Einige wichtige Prinzipien

- Genauigkeit und formales Vorgehen
- Trennung der Verantwortlichkeiten (Separations of concerns)
- Modularität
- Abstraktion
- Voraussehen von Veränderungen (Design for change)
- Allgemeinheit
- Schrittweises Entwickeln (Incrementality)

Genauigkeit und formales Vorgehen

(Unerreichbares) Ideal: Mathematische Genauigkeit

$$
\begin{array}{l}
\{i_1 > 0 \text{ and } i_2 > 0\} \\
\ P \\
\{ \\
\ (\text{exists } z_1, z_2\ (i_1 = o \cdot z_1 \text{ and } i_2 = o \cdot z_2) \\
\ \text{and not} \\
\ (\text{exists } h(\ \text{exists } z_1, z_2 \\
\ (i_1 = h \cdot z_1 \text{ and } i_2 = h \cdot z_2) \text{ and } h > o)) \\
\ \}
\end{array}
$$

Erreichbare Ziele

- klar definierte Regeln
- klar definierte Prozesse
- eindeutig definierte Anforderungen
- rigorose Tests

Trennung der Verantwortlichkeiten

- Verschiedene Probleme separat angehen "Divide and Conquer"
- Reduziert Komplexität jeder Task
- Aufgaben und Verantwortlichkeiten können verteilt und parallelisiert werden

Produkt

Anforderungen separat betrachten

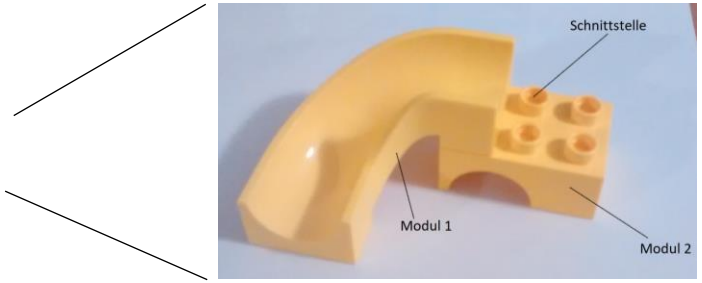
- Funktionalität
- Performance
- Benutzeroberfläche
- ...

Prozess

- Separiere Software Testing und Entwicklung
  - Eigenes Team?
- Phasen in Wasserfallmodell

Modularität

- Komplexes System wird in kleine Teile zerlegt
- Wichtigstes Prinzip in der Softwareentwicklung
- Reduziert Komplexität
  - Erlaubt Trennung der Verantwortlichkeiten



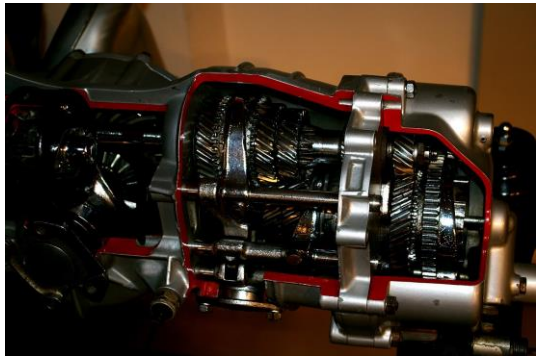
Abstraktion

- Wichtige Aspekte eines Problems werden identifiziert
- Details werden ignoriert
- Spezialfall von "Trennung der Verantwortlichkeiten"
  - wichtig (Funktion)
  - unwichtig (Detail)

Beispiele:

- Programmiersprache (Java)
  - OutputStream
- Betriebssystem
  - Dateien
  - Netzwerk Sockets
- Hardware:
  - Blöcke und Addressen

Abstraktion



Implementation

Allgemeinheit

Leitfrage: Gibt es ein generelleres Problem gibt, welches sich hinter einem Problem versteckt?

Spezifisch

```
public interface IntList {
        List<String> map(Function<Integer, String> mapper) {
            List<String> l = new LinkedList<String>();
            for (int element : list) {
                l.add(mapper(element));
            }
            return l;
        }
    }
```

Allgemein

```
public interface List<T> {
        <R> List<R> map(Function<? super T, ? extends R> mapper) {
            List<R> l = new LinkedList<R>();
            for (T element : list) {
                l.add(mapper(element));
            }
            return l;
        }
    }
```

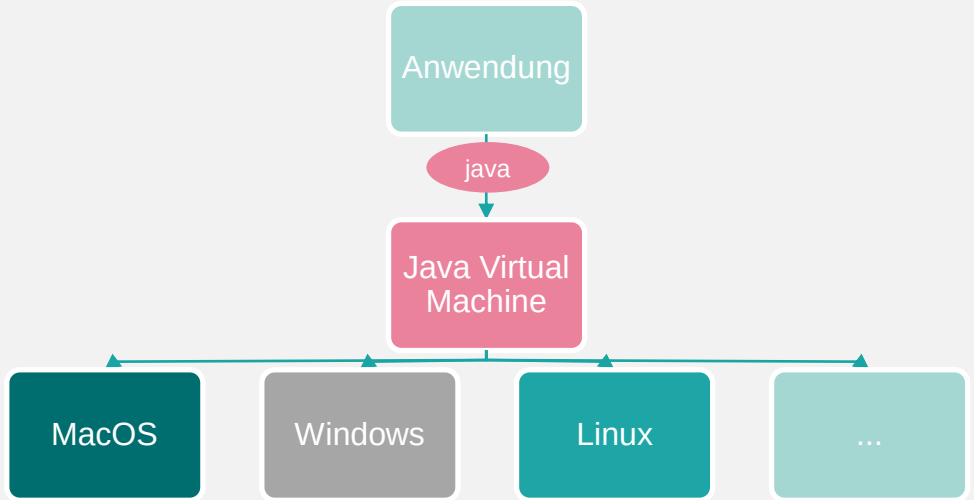
Veränderungen voraussehen

Was wird sich wahrscheinlich ändern?

- Neue Gesetze
- Neue Technologien
- Neue Benutzergruppe
- …

Beispiel: Design von Java:

- Änderungen in OS warden von JVM abstrahiert



Schrittweises Entwickeln

Problem wird inkrementell gelöst

- “growing” statt “building”
- Erstes Resultat nur Approximation von Endprodukt
- Aber: in jedem Inkrement wird etwas Brauchbares produziert

Diskussion - Reflexion

No Silver Bullet

The first step toward the management of disease was replacement of demon theories by germ theory. That very step, the beginning of hope, in itself dahsed all hopes of magial solutions. It told workers that progress would be made stepwise, at great efforts [...] and unremitting care would have to be paid to a discipline of cleanliness.

Although we see no startling breakthroughts, and indeed, believe such to be inconsistent with the nature of software, many encouraging innovations are under way. A disciplined, consistent effort to develop, propagate, and exploit them should indeed yield an order-of-magnitude improvement.

There is no royal road, but there is a road.

Diskussion: No Silver Bullet

- Welche Punkte fanden Sie besonders Bemerkswert oder möchten Sie diskutieren?

- Geben Sie ein Beispiel von *accidental* und *essential complexity* aus Ihrem Lehreralltag:

- Brooks sagt, dass Komplexität der Ursprung vieler Probleme ist. Als Beispiele nennt er:
  - Kommunikation zwischen Teammitgliedern wird schwierig
  - Da Funktionen komplex sind, ist auch deren Aufruf Komplex
  - Komplexe Strukturen sind schwierig zu erweitern

  Warum ist Komplexität so schlimm? Weshalb kommt diese Komplexität in Bauprojekten nicht/weniger zum Vorschein?

Modularität

Was ist ein Modul

Ein Modul ist ein wohldefinierter Teil einer Software

Beispiele

- Eine Funktion/Methode
- Eine Sammlung von Funktionen/Methoden
- Eine Sammlung von Daten
- Eine Klasse / Ein Objekt
- Ein Package

```
public final class Math {
    public static final double E = 2.718281828459045235 4;
    public static double sin(double a) { /* …*/ }
    public static double tan(double a) { /* …*/ }
    …
}
```

Modularität: Top-down vs bottom up

Modularität erlaubt "Teilen der Verantwortlichkeiten" in zwei Phasen:

Top-down design

1. Ausarbeiten der Details (eines Moduls)
   - Andere Module werden ignoriert

2. Integration in Gesamtes Berücksichtigung der Beziehungen zwischen den Modulen.
   - Vernachlässigung der Details der Module.

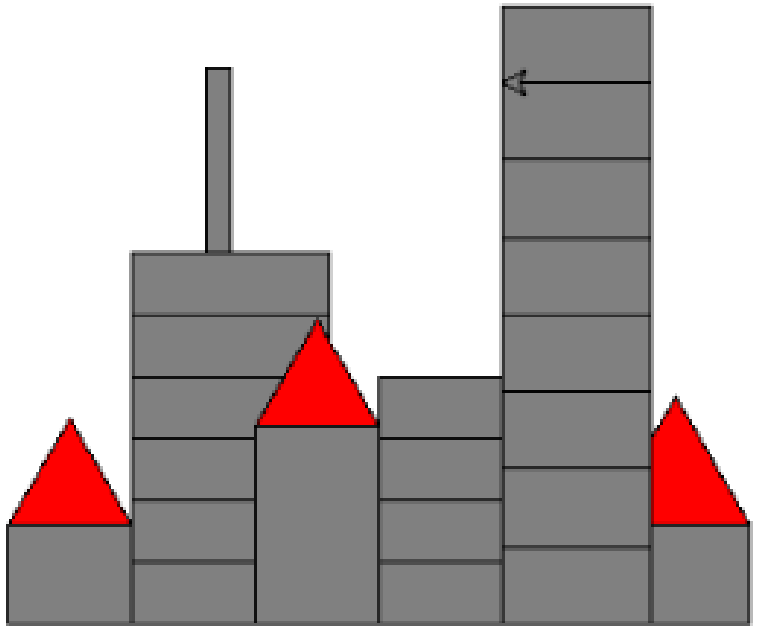
Bottom-up design

Dekomposition und Komposition

Modularität erlaubt, ein System

1. in einfachere Teile zu zerlegen
2. aus einfacheren Teilen zusammenzubauen (Lego)
3. in kleinen Blöcken verstehen zu können.
4. zu verändern, indem nur ein kleiner Teil (ein Modul) verändert wird.

Modularisierung ist im Zentrum der Software-Entwicklung



Formale Definition der Modulbeziehungen

Gegeben eine Menge von Modulen:

$$S = \{M_1, \dots, M_n\}$$

Wir definieren Beziehung r als Relation:

$$r \subset S \times S$$

Wir schreiben:

$$M_i \; r \; M_j \text{ falls } \left(M_i, M_j\right) \in r$$

$\mathrm{M_i r\ M_j}$ bedeutet: Module $\mathrm{M_i}$ steht in Beziehung zu Module $\mathrm{M_j}$

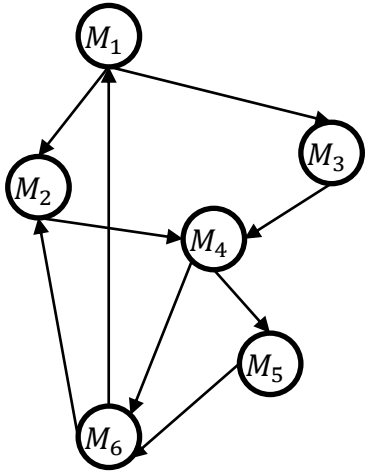
Modulbeziehungen: Beispiel

Module: $S = \{M_1, \dots, M_6\}$

Beziehung: $r \subset \{(M_1, M_1), (M_1, M_2), \dots, (M_6, M_5), (M_6, M_6)\}$

$r = \{(M_1, M_2), (M_1, M_3), (M_2, M_4), (M_3, M_4), (M_4, M_5), (M_4, M_6), (M_5, M_6), (M_6, M_2), (M_6, M_1)\}$

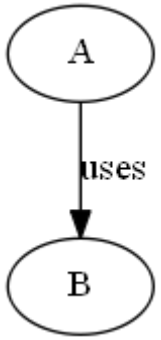
Darstellung als gerichteten Graphen:



Beispielbeziehung 1: “Uses” Beziehung

A "uses" B

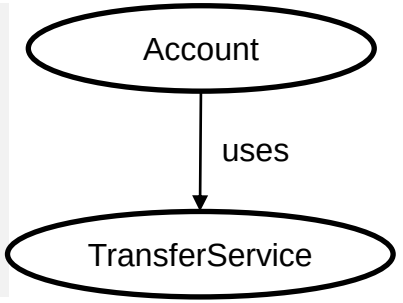
- Module A nutzt Funktionalität von B
- B stellt Funktionalität zur Verfügung



auch

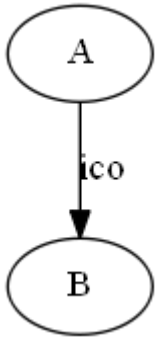
- A ist der "client", B der "server"

```
class Account {
       public static void transferMoney(int amount, Account to) {
           TransferService.transfer(this, to, amount);
       }
}
```

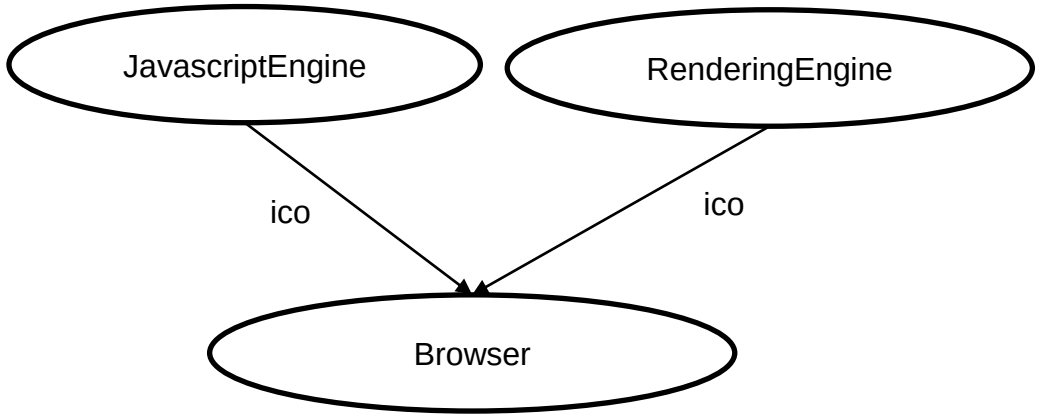


Beispielbeziehung 2, Is component of (Komposition)

- Modul ist aus einfacheren Modulen zusammengesetzt
- Auch Aggregation oder Komposition genannt.
- Beschreibung eines Moduls auf höherer Abstraktionsebene.



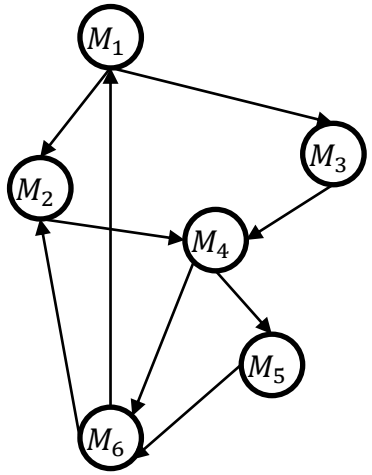

```
class Browser {
	JavaScriptEngine jsEngine;
	RenderingEngine renderingEngine;

	void renderHTML(HTML doc) {
		jsEngine.executeJS(doc);
		renderingEngine.render(doc);
	}
}
```



Modulbeziehungen: Transitive Hülle

Definition der Transitiven Hülle r^+:

$M_1 r^+ M_2$ genau dann, wenn
$M_i r\ M_j$
oder
$\exists M_k \in S$, so dass $M_i r M_k$ und $M_k r^+ M_j$



Modulbeziehungen: Hierarchien und Ebenen

Definition Hierarchie

Eine binäre Relation r ist eine Hierarchie genau dann, wenn

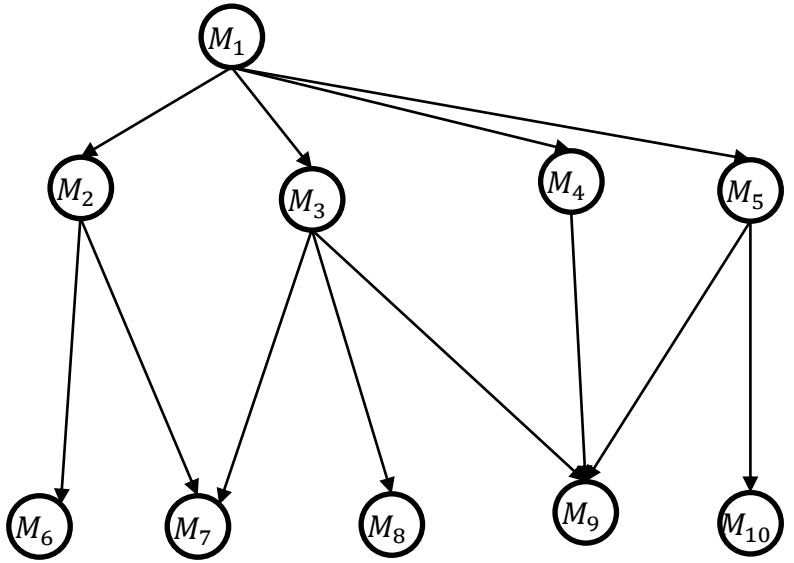
$\nexists(M_i, M_j)$, *so dass* $M_i r^+ M_j$ *und* $M_j r^+ M\_i$

Es gibt keinen Zyklen im Graph

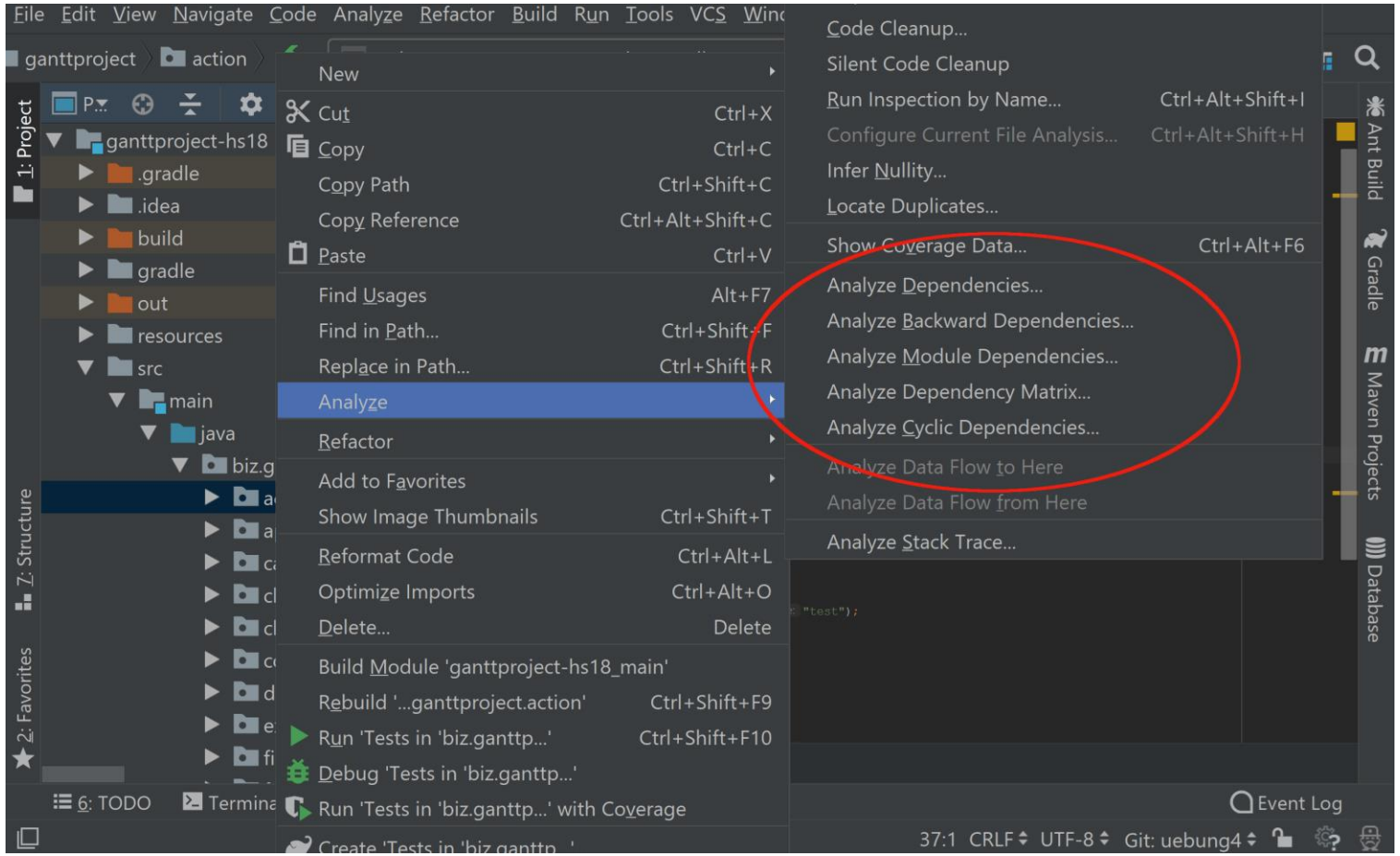
Definition Ebene

M_i ist auf Ebene 0, falls $\nexists M_j$, so dass $M_i r M_j$

- Sei k die höchste Ebene aller Module M_j, so dass $M_i r\ M_j$. Dann ist M_i auf Ebene $k+1$



Praktische Anwendung von Uses Graphen (1)



Praktische Anwendung von Uses Graphen (2)