Advanced Software Engineering 2

- Advanced Software Engineering 2
  - Programmierprinzipien
    - ToDo
    - Solid
      - (S)ingle Responsibility
      - (O)pen Closed Principle
      - (L)iskov Substitution Principle
      - (I)nterface Segregation Principle
      - (D)ependency Inversion Principle
    - Tell, don’t ask
    - Kiss (Keep it simple, stupid)
    - SLAP (Single Level of Abstraction Principle)
    - GRASP
    - Low Coupling
    - High Cohesion
    - Information Expert
    - Indirection
    - DRY (Don’t Repeat Yourself !)
  - DevOps
    - Warum
    - Lean

Programmierprinzipien

- sind Leitfaden
- Verantwortung festlegen

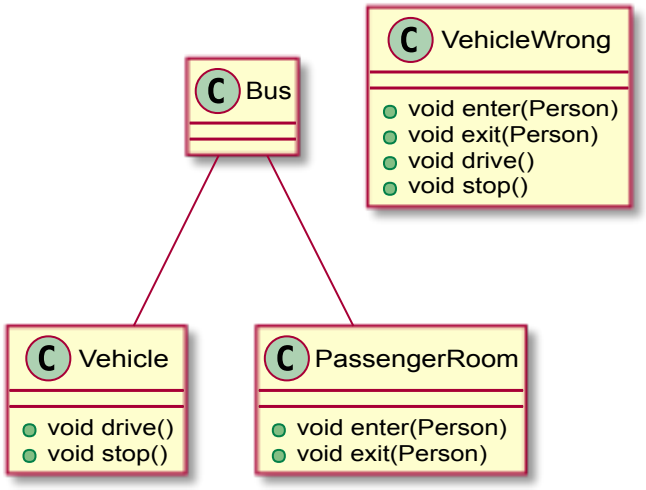
ToDo

- [ ] Ergänze fehlende Prinzipien
- [ ] Füge Graphen hinzu
- [ ] Erkläre kurz

Solid

(S)ingle Responsibility

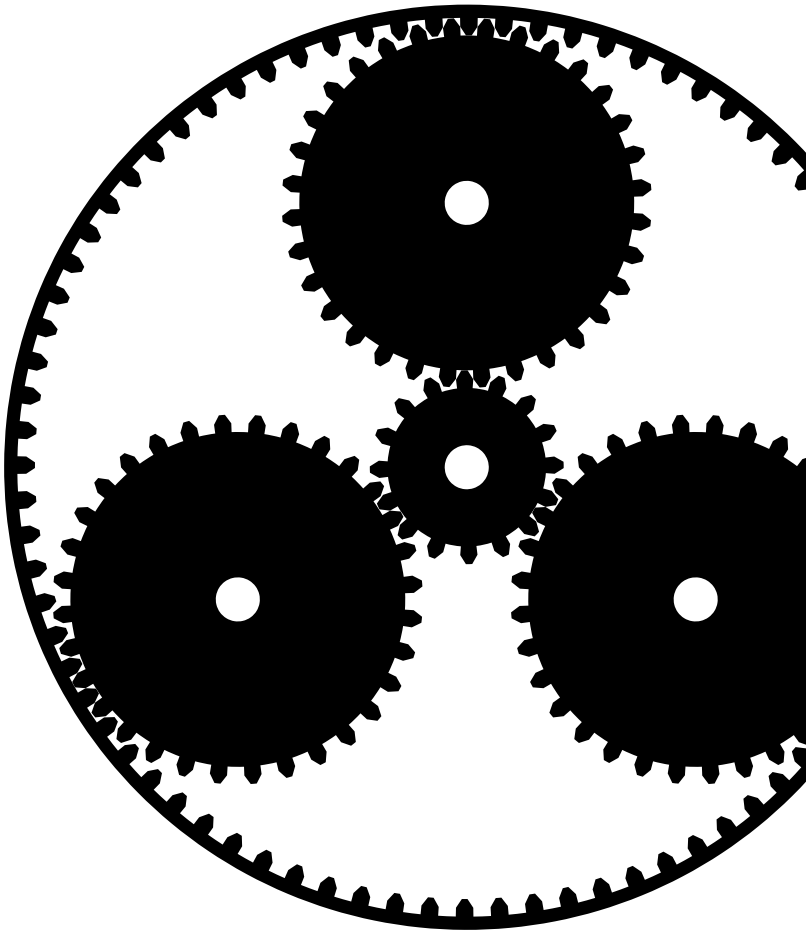
- Klasse sollte nur einen Grund oder Ursache haben, sich zu ändern
- jede Klasse nur eine Zuständigkeit
- eine Klasse erhält Achsen, auf der sich Anforderungen ändern können
    - jede Zuständigkeit-> neue Achse, nur eine Achse pro Klasse



```
var width = 960,
    height = 500,
    radius = 80,
    x = Math.sin(2 * Math.PI / 3),
    y = Math.cos(2 * Math.PI / 3);

var offset = 0,
    speed = 4,
    start = Date.now();

var svg = d3.select("#d3-example")
    .attr("width", width)
    .attr("height", height)
  .append("g")
    .attr("transform", "translate(" + width / 2 + "," + height / 2 + ")sc
ale(.55)")
  .append("g");

var frame = svg.append("g")
    .datum({radius: Infinity});

frame.append("g")
    .attr("class", "annulus")
    .datum({teeth: 80, radius: -radius * 5, annulus: true})
  .append("path")
    .attr("d", gear);

frame.append("g")
    .attr("class", "sun")
    .datum({teeth: 16, radius: radius})
  .append("path")
    .attr("d", gear);

frame.append("g")
    .attr("class", "planet")
    .attr("transform", "translate(0,-" + radius * 3 + ")")
    .datum({teeth: 32, radius: -radius * 2})
  .append("path")
    .attr("d", gear);

frame.append("g")
    .attr("class", "planet")
    .attr("transform", "translate(" + -radius * 3 * x + "," + -radius * 3 *
 y + ")")
    .datum({teeth: 32, radius: -radius * 2})
  .append("path")
    .attr("d", gear);

frame.append("g")
    .attr("class", "planet")
```

```
    .attr("transform", "translate(" + radius * 3 * x + "," + -radius * 3 *
+ ")")
    .datum({teeth: 32, radius: -radius * 2})
  .append("path")
    .attr("d", gear);

d3.selectAll("input[name=reference]")
  .data([radius * 5, Infinity, -radius])
    .on("change", function(radius1) {
      var radius0 = frame.datum().radius, angle = (Date.now() - start) * sp
ed;
      frame.datum({radius: radius1});
      svg.attr("transform", "rotate(" + (offset += angle / radius0 - angl
e / radius1) + ")");
    });

d3.selectAll("input[name=speed]")
    .on("change", function() { speed = +this.value; });

function gear(d) {
  var n = d.teeth,
      r2 = Math.abs(d.radius),
      r0 = r2 - 8,
      r1 = r2 + 8,
      r3 = d.annulus ? (r3 = r0, r0 = r1, r1 = r3, r2 + 20) : 20,
      da = Math.PI / n,
      a0 = -Math.PI / 2 + (d.annulus ? Math.PI / n : 0),
      i = -1,
      path = ["M", r0 * Math.cos(a0), ",", r0 * Math.sin(a0)];
  while (++i < n) path.push(
      "A", r0, ",", r0, " 0 0,1 ", r0 * Math.cos(a0 += da), ",", r0 * Mat
h.sin(a0),
      "L", r2 * Math.cos(a0), ",", r2 * Math.sin(a0),
      "L", r1 * Math.cos(a0 += da / 3), ",", r1 * Math.sin(a0),
      "A", r1, ",", r1, " 0 0,1 ", r1 * Math.cos(a0 += da / 3), ",", r1 * M
th.sin(a0),
      "L", r2 * Math.cos(a0 += da / 3), ",", r2 * Math.sin(a0),
      "L", r0 * Math.cos(a0), ",", r0 * Math.sin(a0));
  path.push("M0,", -r3, "A", r3, ",", r3, " 0 0,0 0,", r3, "A", r3, ",", r3
 " 0 0,0 0,", -r3, "Z");
  return path.join("");
}

d3.timer(function() {
  var angle = (Date.now() - start) * speed,
      transform = function(d) { return "rotate(" + angle / d.radius + ")";
};
  frame.selectAll("path").attr("transform", transform);
  frame.attr("transform", transform); // frame of reference
});
```

```
[object Object]
```



(O)pen Closed Principle

Elemente der Software wie Klassen, Module und Funktionen sollten

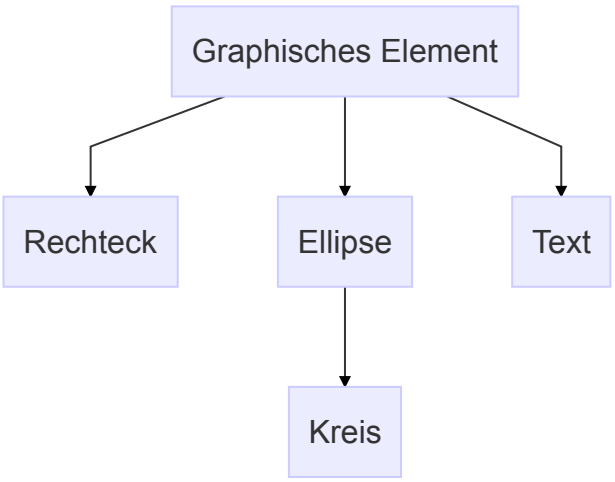
- offen für Erweiterung sein
- geschlossen für Änderungen sein

Erweiterung nur über Vererbung (optimal)

(L)iskov Substitution Principle

- Abgeleitete Typen müssen schwächere Vorbedingungen haben
- Abgeleitete Typen müssen stärkere Nachbedingungen haben

> Wenn sich das Objekt so verhält, wie sein Oberklasse



(I)nterface Segregation Principle

Anwender sollen nicht von Funktionen abhängig sein, die sie nicht brauchen

> Übergebe User nur Interface mit Funktionen, die er benötigt

(D)ependency Inversion Principle

High-Level Module von Low-Level Modulen abhängig

- schlecht
- besser : High-Level Modul von Abstraktionen abhängig

Abstraktionen : nicht von Details abhängig, Details abhängig von Abstraktion

Tell, don’t ask

- Prozeduraler Code kappelt sich stark an andere Elemente

Kiss (Keep it simple, stupid)

Herkunft in der US Navy 1960

- einfache Systeme arbeiten am besten
- Komplexität unter allen Umständen vermeiden
- Linux Arch

SLAP (Single Level of Abstraction Principle)

- Prinzip des einfachen Abstraktionsniveau
- keine Vermischung von Arbeit und Delegation
- keine Vermischung von DB und Businesslogik
- Fördert Wiederverwendbarkeit

GRASP

Low Coupling

Die Abhängigkeiten verringern, um das eigene Paket von der Landschaft “abzukoppeln”

> Koppelung ist der Maß von Abhängigkeiten von Paketen und Objekten

Effekte:

- geringe Abhängigkeiten zu Änderungen in anderen Teilen
- einfacher testbar
- verständlicher, da weniger Kontext notwendig ist
- einfacher wiederverwendbar

High Cohesion

Kohäsion ist ein Maß für die Zusammenhalt einer Klasse. Hohe Kohäsion und lose Kopplung als Fundament für idealen Code.

Information Expert

Kapselung von Informationen
Leichtere Klassen, da Businesslogik zu den Daten verteilt wird

Indirection

DRY (Don’t Repeat Yourself !)

- wiederhole dich nicht
- Anwendbar:
- Datenbankschema
- Testpläne
- Buildsystem
- Dokumentation
- Gegenteil:

- WETYAGNI (You ain’t gonna need it)du wirst es nicht brauchen

DevOps

- eine **Bewegung** mit dem Ziel **Time-To-Market** einer **Änderungseinheit** zu reduzieren, bei gleichzeitiger Gewährleistung **hoher Qualität**
- durch Anwendung des **Lean-Prinzip** auf den gesamten **Software-Wertstrom**

Warum

- Dev **schnell Veränderungen umsetzen**
- Ops (Administrator) sollen **Sicherheit und Stabilität** der Systeme gewährleisten
-

Lean

Philsophie, mit Ziel, einen **Prozess** durch die **Eliminierung von Verschwendung kontinuierlich** zu verbessern und dabei die **Bedürfnisse der Kunden** als **Ausgangspunkt allen Handelns** sieht

Verschwendung erkennen:

- Materialbewegung
- Bestände::
- Bewegung
- Wartezeiten
- Verarbeitung
- Überproduktion
- Korrekturen und Fehler

Verschwendung beseitigen:

Pull Prinzip:

- es wird nur produziert:
  - **was** der Kunde will
  - **wenn** der Kunde es will