

## Fragen, Antworten, Kommentare zur aktuellen Vorlesung

Evaluationsergebnisse am Ende des Dokuments.

Hinweis: Unter <https://youtu.be/by9k47IUIAU> (42:23) finden Sie die Erklärung einer Beispiellösung zu den Teilaufgaben `alleWerteGleichOft(.)` und `gleicheWerte(.)` der Messreihe mit Erklärungen zu möglichen Ansätzen.

Es handelt sich nebenbei um keine Musterlösungen, da bereits jetzt eine Komplexität erreicht wird, dass es verschiedene, teilweise gleich gute Lösungen gibt. Ein konkretes Beispiel ist die individuelle Überprüfung von null-werten. (Das hat für Lehrende nebenbei den positiven Seiteneffekt, dass Lösungskopien schnell erkannt werden). Musterlösungen haben zwei negative Effekte. Zunächst wie angedeutet, dass kreative Lösungen eventuell nicht als gut erkannt werden. Ein zweiter Effekt ist ein Sammlungseffekt bei Studierenden und sie nur gestapelt werden, da gehofft wird, die Aufgaben am Ende schon zu verstehen. Bedenken Sie immer, dass der Schritt von der Fähigkeit ein Programm zu lesen, die fast alle erreichen zum selbständigen Schreiben von Programmen ein riesengroßer ist, der nur durch intensive individuelle Übung machbar wird.

In den Lösungen zeige ich einen systematischen Ansatz und versuche möglichst unsere Coding Guidelines zu erfüllen, die auf der Veranstaltungsseite stehen. Mit etwas mehr Programmierer- und Software-Engineering-Erfahrungen würden die Lösungen teilweise anders aussehen, aber das ist Thema Ihrer nächsten Semester.

Denken Sie daran, dass es immer Möglichkeiten gibt, über Ihre individuellen Lösungen zu diskutieren. Dies ist u. a. Teil des zweiten Praktikumstermins. Sie können aber auch zur Vorlesungszeit vorbeischaun oder mir eine E-Mail schreiben. Meine Betreuungszeit ist während der Vorlesungszeit aktuell noch nicht ausgelastet.

Hinweis: Ein gerne gemachter objektorientierter Fehler wird hier am Beispiel von `gleicheWerte(.)` gezeigt, dabei wird eine bereits vorher geschriebene Hilfsmethode `vorkommenVon()` genutzt, die die Anzahl der Vorkommen eines Wertes zählt.

```
public int vorkommenVon(Messreihe mr, int wert) { // sehr schlecht!!!
    if (mr == null || mr.getMesswerte() == null) {
        return 0;
    }
    int ergebnis = 0;
    for (int i: mr.getMesswerte()) {
        if (i == wert) {
            ergebnis = ergebnis + 1;
        }
    }
    return ergebnis;
}

public boolean gleicheWerte(Messreihe m) {
    if (this.messwerte == null || this.messwerte.isEmpty()) { // oder .size() == 0
        return m == null || m.getMesswerte() == null
            || m.getMesswerte().isEmpty();
    }
    if (m == null || m.getMesswerte() == null) {
```

```

    return false;
}
for (int i: this.messwerte) {
    if (m.vorkommenVon(m, i) == 0) {
        return false;
    }
}
for (int i: m.getMesswerte()) {
    if (this.vorkommenVon(this, i) == 0) {
        return false;
    }
}
return true;
}

```

Generell läuft die Lösung, aber der Ansatz der Methode `vorkommenVon()` ist falsch, da hier als erster Parameter eine Messreihe (genauso schlecht eine `ArrayList`) übergeben wird. Wir haben damit in der Klasse `Messreihe` eine Methode geschrieben, die eine andere Messreihe verarbeiten soll. Betrachtet man die Idee von `vorkommenVon()` aber genauer, bezieht sich der Aufruf immer genau auf die Messreihe, deren Methode aufgerufen wird. Daraus folgt das der Parameter hier überflüssig ist. Aus Sicht der Objektorientierung ist das ein massiver Fehler, den man genau einmal machen darf. Die Korrekturen sehen wie folgt aus.

```

public int vorkommenVon(int wert) {
    if (this.messwerte == null) {
        return 0;
    }
    int ergebnis = 0;
    for (int i: this.messwerte) {
        if (i == wert) {
            ergebnis = ergebnis + 1;
        }
    }
    return ergebnis;
}

public boolean gleicheWerte(Messreihe m) {
    if (this.messwerte == null || this.messwerte.isEmpty()) {
        return m == null || m.getMesswerte() == null
            || m.getMesswerte().isEmpty();
    }
    if (m == null || m.getMesswerte() == null) {
        return false;
    }
    for (int i: this.messwerte) {
        if (m.vorkommenVon(i) == 0) {
            return false;
        }
    }
    for (int i: m.getMesswerte()) {
        if (this.vorkommenVon(i) == 0) {
            return false;
        }
    }
    return true;
}

```

Hinweis: Nutzt man KI zu Generierung von Ergebnissen, um diese z. B. mit seinen eigenen Lösungen zu vergleichen, macht die KI häufig auch obigen Fehler. Eine Lösung von Grok sieht wie folgt aus:

```

private ArrayList<Integer> getUnique(ArrayList<Integer> list) {
    ArrayList<Integer> unique = new ArrayList<>();
    if (list == null) {
        return unique;
    }
    for (int i = 0; i < list.size(); i++) {
        int val = list.get(i);
        boolean exists = false;
        for (int j = 0; j < unique.size(); j++) {
            if (unique.get(j) == val) {
                exists = true;
                break;
            }
        }
        if (!exists) {
            unique.add(val);
        }
    }
    return unique;
}

private int anzahl(int wert, ArrayList<Integer> list) {
    if (list == null) {
        return 0;
    }
    int count = 0;
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i) == wert) {
            count++;
        }
    }
    return count;
}

public boolean alleWerteGleichOft(Messreihe other) {
    ArrayList<Integer> list1 = (messwerte == null)
        ? new ArrayList<>() : messwerte;
    ArrayList<Integer> list2 = (other == null || other.messwerte == null)
        ? new ArrayList<>() : other.messwerte;
    ArrayList<Integer> u1 = getUnique(list1);
    ArrayList<Integer> u2 = getUnique(list2);
    ArrayList<Integer> combined = new ArrayList<>();
    for (Integer val : u1) {
        boolean exists = false;
        for (Integer c : combined) {
            if (c.equals(val)) {
                exists = true;
                break;
            }
        }
        if (!exists) {
            combined.add(val);
        }
    }
    for (Integer val : u2) {
        boolean exists = false;
        for (Integer c : combined) {
            if (c.equals(val)) {
                exists = true;
                break;
            }
        }
    }
}

```

```

    }
    if (!exists) {
        combined.add(val);
    }
}
for (Integer val : combined) {
    if (anzahl(val, list1) != anzahl(val, list2)) {
        return false;
    }
}
return true;
}

```

Neben dem massiven OO-Fehler sollte ebenfalls die unnötig komplexe Programmierung auffallen. Es ist auch unwahrscheinlich, dass Erstis „?:“ nutzen. Es ist weiterhin unwahrscheinlich und unsauber „break“ außerhalb von switch-case zu nutzen.

Auf Nachfrage erkennt die KI zumindest Ihren Fehler:

Mein Prof hat mich durchfallen lassen, da bei Aufgabe 28 die Methode `private int anzahl(int wert, ArrayList<Integer> list)` angeblich nicht objektorientiert ist.

## Erklärung zum Problem

Dein Professor hat recht: Die Methode `private int anzahl(int wert, ArrayList<Integer> list)` (und ähnlich `getUnique`) ist nicht objektorientiert (OO), weil sie wie eine statische Hilfsfunktion wirkt, die auf einer externen `ArrayList` operiert, anstatt auf den Instanzdaten der Klasse (`this.messwerte`). In der OO-Programmierung (insbesondere in Java, wie in deiner Vorlesung VL 15 betont) sollten Methoden der Klasse auf ihren eigenen Attributen arbeiten, um Kapselung und Objektverantwortung zu wahren. Das bedeutet:

- **Nicht OO:** Die Methode nimmt eine externe Liste als Parameter und zählt darin – das könnte genauso gut eine statische Methode in einer Utility-Klasse sein. Es verletzt das Prinzip, dass ein Objekt für seine eigenen Daten verantwortlich ist.
- **OO-konform:** Mache die Methode zu einer Instanzmethode, die direkt auf `this.messwerte` zugreift. Für Vergleiche mit einer anderen Messreihe (z.B. in `alleWerteGleichOft`) rufst du einfach `other.anzahl(wert)` auf – so nutzt jedes Objekt seine eigene Methode.

Aus dem Aufgabenblatt (basierend auf den Screenshots):

- Die Aufgabe fordert "Klausurähnlichkeit" und "klare Algorithmen" mit Schleifen (VL 15).
- Tests prüfen u.a. auf unveränderte Listen (`messwerteUnveraendert`), Null-Handling und korrekte Berechnungen.
- Keine externen Collections wie Set oder Map erlaubt – nur Schleifen und `ArrayList`-Methoden (`add`, `get`, `size`, `iterator` usw.).
- Die Anmerkung zur "selbständigen Lösung" und "Ideen der Algorithmen" unterstreicht, dass der Code klar und OO sein muss, ohne unnötige Parameter.

Deine Lösung funktioniert technisch (Tests bestehen), aber der Stil ist nicht OO. Ich habe die Klasse korrigiert:

- anzahl und getUnique zu Instanzmethoden ohne List-Parameter gemacht.
- In Vergleichsmethoden (alleWerteGleichOft, gleicheWerte, verschieden) lokale Variablen für Null-Handling verwendet und die Methoden der jeweiligen Instanz aufgerufen.
- Keine Änderung an Funktionalität oder Tests – nur OO-Verbesserung.
- Begründung pro Methode in Kommentaren.

## Korrigierte Klasse Messreihe

```
// Korrigierte OO-Version: Operiert auf this.messwerte,
// kein externer Parameter
// Begründung: Instanzmethode, die eigene Daten verarbeitet;
// für other ruf other.anzahl() auf
private int anzahl(int wert) {
    if (this.messwerte == null) {
        return 0;
    }
    int count = 0;
    for (int i = 0; i < this.messwerte.size(); i++) {
        if (this.messwerte.get(i) == wert) {
            count++;
        }
    }
    return count;
}
}
```

...

Wenn sich jemand fragt, warum die KI zunächst eine ineffiziente und unsaubere Lösung vorschlägt muss man sich daran erinnern, wie eine KI lernt. Da die Messreihe keine 100%-Standardaufgabe ist, wird sie auf eine absolute Standardaufgabe, die Analyse zweier Listen, reduziert. Diese Algorithmen werden oft nicht im objektorientierten Kontext erklärt, weshalb die KI diesen nicht als Standard ansieht. Oftmals wird „Instanz“ etwas unsauber statt „Objekt“ genutzt, was auf einem Übersetzungsproblem von „instance“ basiert.

Frage: Gibt es eine Möglichkeit systematisch zu analysieren, auf welche Nullwerte geachtet werden muss?

Antwort: Grundsätzlich ist anzunehmen, dass alle übergebenen Objekte, also nicht elementare Datentypen, null sein können. Dies kann ebenfalls auf alle Objekte zutreffen die in Objektvariablen stehen. Daraus kann eine Boolesche Matrix gebaut und dann für einzelne Fälle, meist Gruppen von Fällen, entschieden werden.

Beispiel: Klasse Messreihe;

```
public class Messreihe {
    private String messort;
    private ArrayList<Integer> messwerte;

    public boolean gleicheWerte(Messreihe other) {
        // hier soll in diesem Fall fuer das Ergebnis null
        // wie eine leere Liste behandelt werden
    }
}
```

In der Methode können `this.messwerte`, `this.messort`, `other`, `other.messwerte` und `other.messort` null sein. Da der `messort` in der Aufgabe nicht interessiert, reduziert es sich auf folgende Fälle:

<code>this.messwerte</code>	<code>other</code>	<code>other.messwerte</code>	Beispielergebnis
null	null	null	true (nach Aufgabe)
<del>null</del>	<del>null</del>	<del>not null</del>	ist nicht möglich, wenn other null ist
null	not null	null	true
null	not null	not.null	Ergebnis abhängig von <code>other.messreihe.size()</code> , wenn 0 dann true
not null	null	null	Ergebnis abhängig von <code>this.messreihe.size()</code> , wenn 0 dann true
<del>not null</del>	<del>null</del>	<del>not null</del>	ist nicht möglich, wenn other null ist
not null	not null	null	Ergebnis abhängig von <code>this.messreihe.size()</code> , wenn 0 dann true
not null	not null	not null	Prüfalgorithmus ausführen

eine von vielen Umsetzungsmöglichkeiten ist:

```

if (this.messwerte == null || this.messwerte.isEmpty()) {
    return other == null || other.getMesswerte() == null
        || other.getMesswerte().isEmpty();
}
if (other == null || other.getMesswerte() == null) {
    return false;
}
// "eigentliches" Programm

```

Es gibt auch Ansätze durch die umgebende Programmierung zu verhindern, dass null-Werte vorkommen können. Dazu muss aber auf einfache set-Methoden verzichtet werden, was das Testen erschwert.

Frage: Wann soll man in Java Hilfsvariablen nutzen, die Speicher verbrauchen und wann nicht?

Antwort: Das mit dem Speicher spielt nur noch bei wenigen sehr hardware-nahen Chips mit integriertem Speicher eine große Rolle und wird im TI-Studium im 5. Semester betrachtet. Generell gilt, dass Hilfsvariablen zu nutzen sind, wenn man dadurch sich wiederholende identische Berechnungen vermeiden kann, z. B.

```
while (notFound && this.anzahlObjekte() > 42) { ...
```

Wenn der Methodenaufruf immer das gleiche Ergebnis liefert, ist folgendes besser:

```
int aktuelleAnzahl = this.anzahlObjekte();
while (notFound && aktuelleAnzahl > 42) { ...
```

Da Compiler aber immer einige statische Optimierungen machen, könnten beide Codes gleich performant sein.

Ein zweiter Punkt zur Nutzung von Hilfsvariablen kann die Erhöhung der Lesbarkeit sein, die sich immer auf die Lesbarkeit für andere Personen bezieht. Da kann es sinnvoll sein bei komplexen Berechnungen ein Teilergebnis an eine Hilfsvariable mit sprechendem Namen zuzuweisen, um den Aufbau des Algorithmus damit zu verdeutlichen.

Frage: In set-Methoden kann man ja auch zurückgeben, ob die Änderung erfolgreich war, ist das sinnvoll?

Antwort: Generell kann man bei Methoden mit Rückgabe void darüber nachdenken, was situationsbedingt sinnvoll sein kann. Bei klassischen set-Methoden ist das eher nicht der Fall, da diese auch von Java selbst und anderen Frameworks genau in dieser Form als existent angenommen werden.

Die Antwort ist nicht einfach, da hier auch das Thema Validierung, die Überprüfung der übergebenen Parameter hineinspielt. Diese Überprüfung kann sehr wohl in set-Methoden erfolgen, was dann auch voraussetzt, dass jedwede Änderung der betroffenen Variable nur über diese set-Methode passieren. Das ist ein sinnvoller Ansatz, wenn er konsequent im Projekt genutzt wird. Statt einem anderen Rückgabewert boolean, wird dann oft auch beim Scheitern der Überprüfung eine Exception geworfen (später in der VL). Es gibt weitere Validierungsansätze z. B. mit Validation-Frameworks.

Frage: Gibt es Regeln wie man die Methoden im Java-Code anordnen soll?

Antwort: Es gibt keine klaren Vorgaben, diese werden aber oft in Projekten oder Unternehmen gemacht. Die Regel erst Konstruktor, dann get, set, hashCode, equals und dann der Rest ist weit verbreitet, da man an diesen Methoden sofort erkennt, ob die übliche Nutzung der Methoden vorliegt. Oft wird aber auch genutzt: Konstruktor, dann Methoden thematisch sortiert, dann Kommentar in Linienform und danach get, set, hashCode, equals, da so alles Interessante direkt ins Auge springt. Da wohl alle Entwicklungsumgebungen für größere Projekte eine Kurzübersicht mit anklickbaren Methodennamen haben, ist die Anordnung nicht so wichtig, wichtig bleiben aber immer die sprechenden Namen.

## **Kommentierte Evaluationsergebnisse**

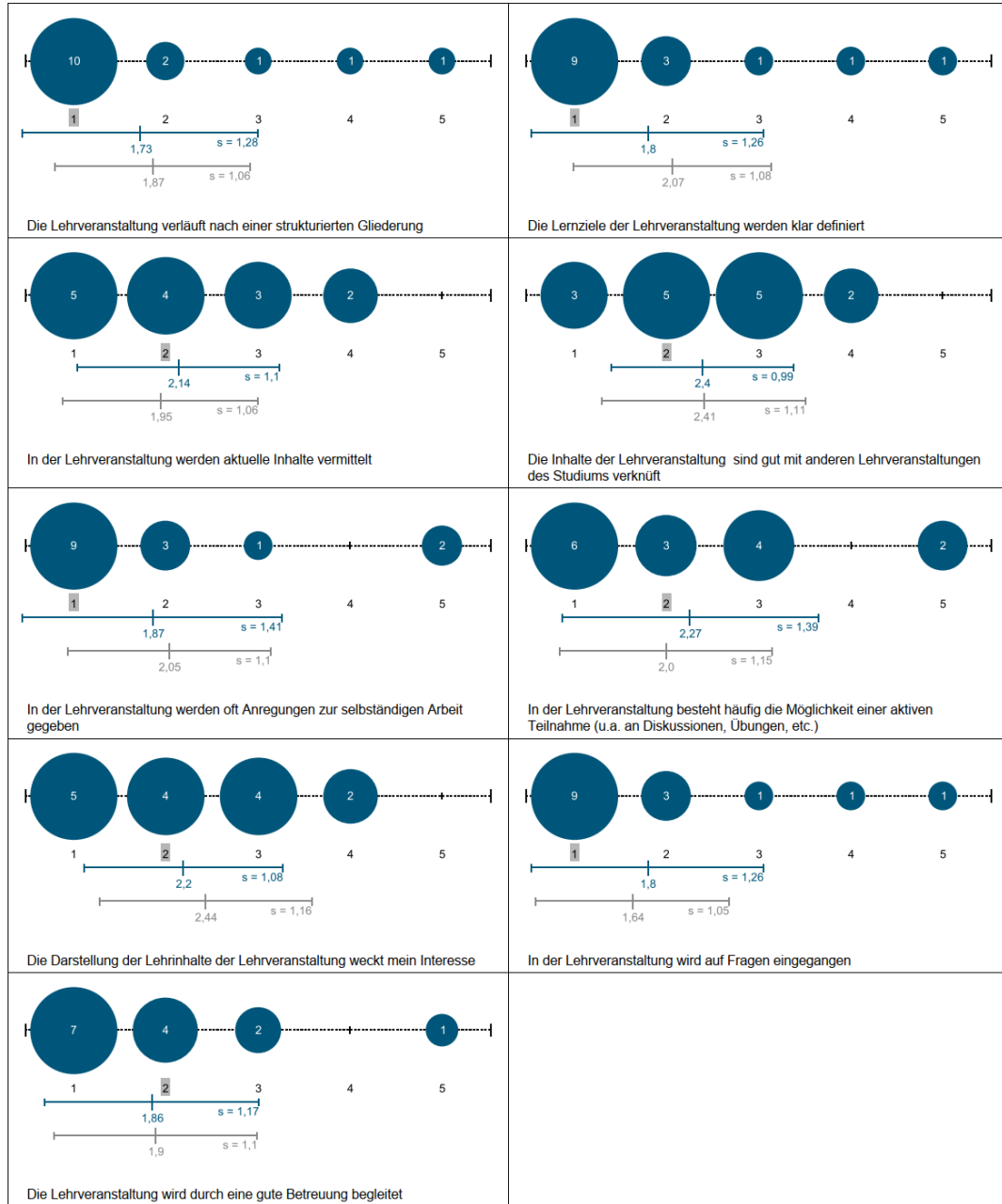
Zunächst Danke für die ordentliche Teilnahme.

Generell ein Kommentar von mir vorweg, der etwas schwer zu formulieren ist. Es fällt auf, dass 2 Studis der Veranstaltung sehr kritisch gegenüber sind, was ich bei einigen Punkten nachvollziehen kann (aber teilweise anders sehe). Mein Punkt ist eher, dass diese Personen anscheinend keine Abschlusskommentare hinterlassen haben. Das macht es für mich schwer nachzuvollziehen, ob es sich um präzise formulierbare Kritik oder um plumpe Pauschalkritik handelt, z. B. „Ich kann gegenderte Texte generell nicht lesen.“ oder „Das Studium gefällt mir generell so nicht“. Bitte, nutzen Sie die Abschlussevaluation, startet nächste Woche, um die fachlichen Punkte zu präzisieren.

## Auswertung zur Veranstaltung Programmierung 1 (I) (Vorlesung)

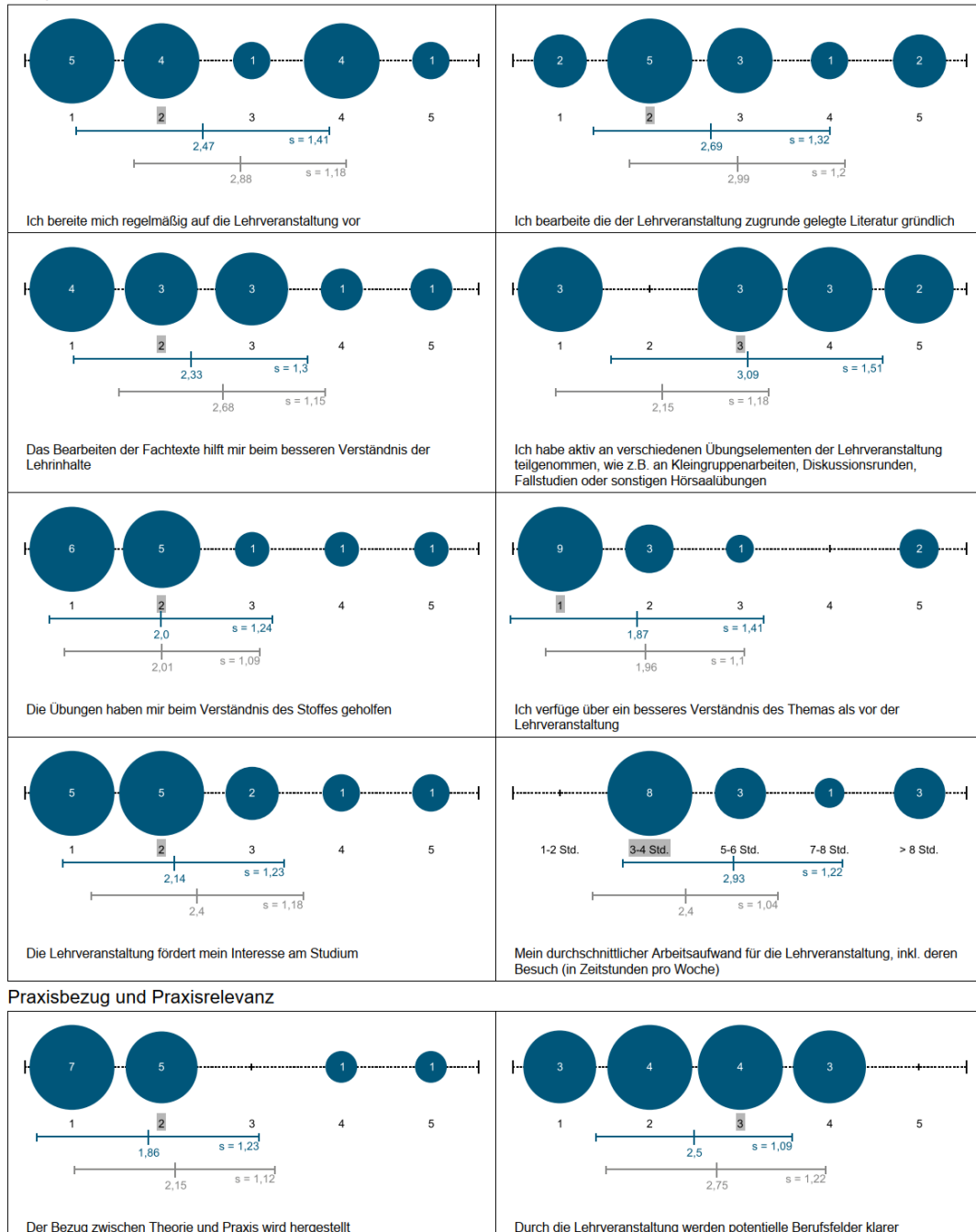
Liebe Lehrende,  
 anbei erhalten Sie die Ergebnisse der Evaluation Ihrer Lehrveranstaltung.  
 Zu dieser Veranstaltung wurden 15 Bewertungen (bei 33 Teilnehmenden) abgegeben. Dies entspricht einer Rücklaufquote von 45%.  
 Erläuterungen zu den Diagrammen befinden sich am Ende dieses Dokuments.  
 Mit freundlichen Grüßen,  
 Das Evaluationsteam

### Lehrprozess



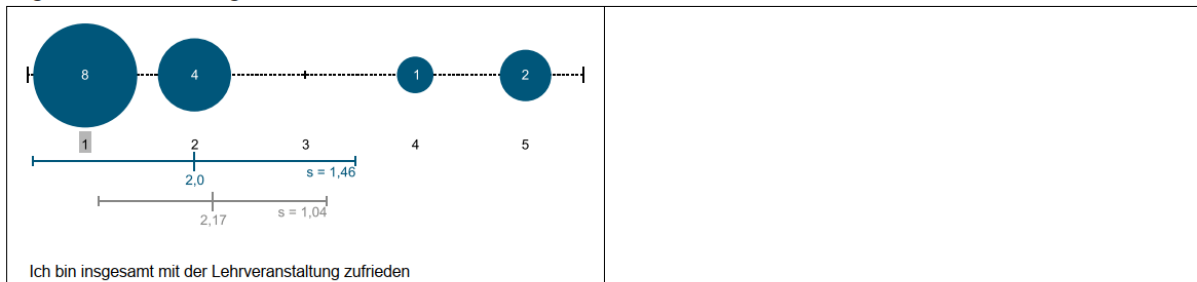
Anmerkung: Da Sie individuell Ihren Lernprozess strukturieren sollen, kann ich bei einzelnen Punkten nicht beurteilen, welche Antworten besser sind. Wichtig ist nur, dass Sie am Ende solide programmieren können.

Lernprozess



Das potenzielle Berufsfeld wird in der Veranstaltung „nur“ im Fundament sichtbar. Sie müssen sehr gut in der Programmierung sein, um erfolgreich in der Informatik zu werden. Es gibt aber Leute, die sehr gut programmieren können, aber trotzdem in IT-Projekten nicht sinnvoll mitarbeiten können.

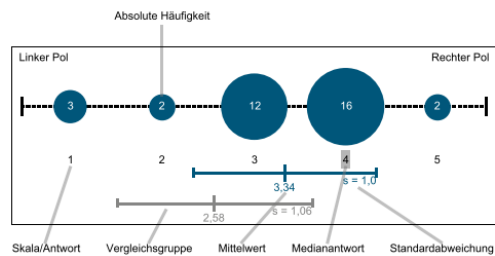
### Allgemeineinschätzung



### Anregungen, Lob und Kritik (IN DRUCKSCHRIFT)

- Das Format mit Online-Videos passt sehr gut zur Programmieren VL, da man die Videos pausieren kann, um selbst Code zu schreiben.
- Die Verfügbarkeit der Vorlesung als Aufzeichnungen ist mir sehr willkommen, der (spontane) Austausch mit dem Professor und anderen Studierenden fehlt mir allerdings schon. Zwar ist der Professor zu den geplanten Vorlesungszeiten und auch sonst gut zu erreichen, ein Bezugs- oder Gemeinschaftsgefühl entsteht bei dieser Art der reinen Online-Vorlesung aber leider nicht.
- Es ist gut das die Veranstaltung online stattfindet
- Ich bin sehr glücklich und zufrieden mit dem Stil und der Struktur dieser Lehrveranstaltung. Der Professor und der Mitarbeiter sind auch alle sehr großartig.

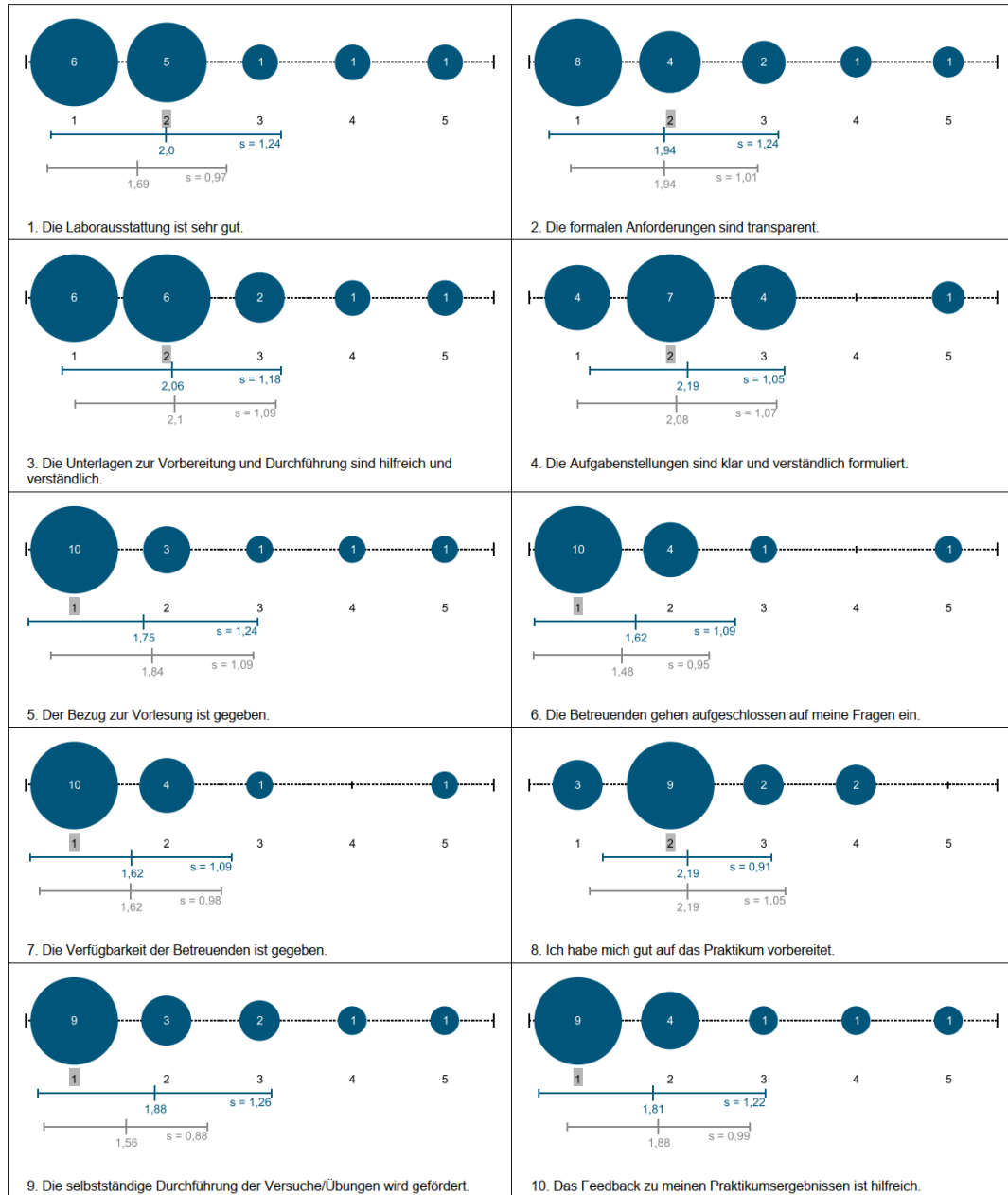
### Legende



Ich halte es auch für sehr wichtig, dass nicht alle Veranstaltungen online sind, um eine eng vernetzte Gruppe aufzubauen. Eine Frage zur sinnvollen Anzahl an Veranstaltungen steht in der Abschlussevaluation.

## Auswertung zur Veranstaltung Programmierung 1 (Praktikum)

Liebe Lehrende,  
 anbei erhalten Sie die Ergebnisse der Evaluation Ihrer Lehrveranstaltung.  
 Zu dieser Veranstaltung wurden 16 Bewertungen abgegeben.  
 Erläuterungen zu den Diagrammen befinden sich am Ende dieses Dokuments.  
 Mit freundlichen Grüßen,  
 Das Evaluationsteam

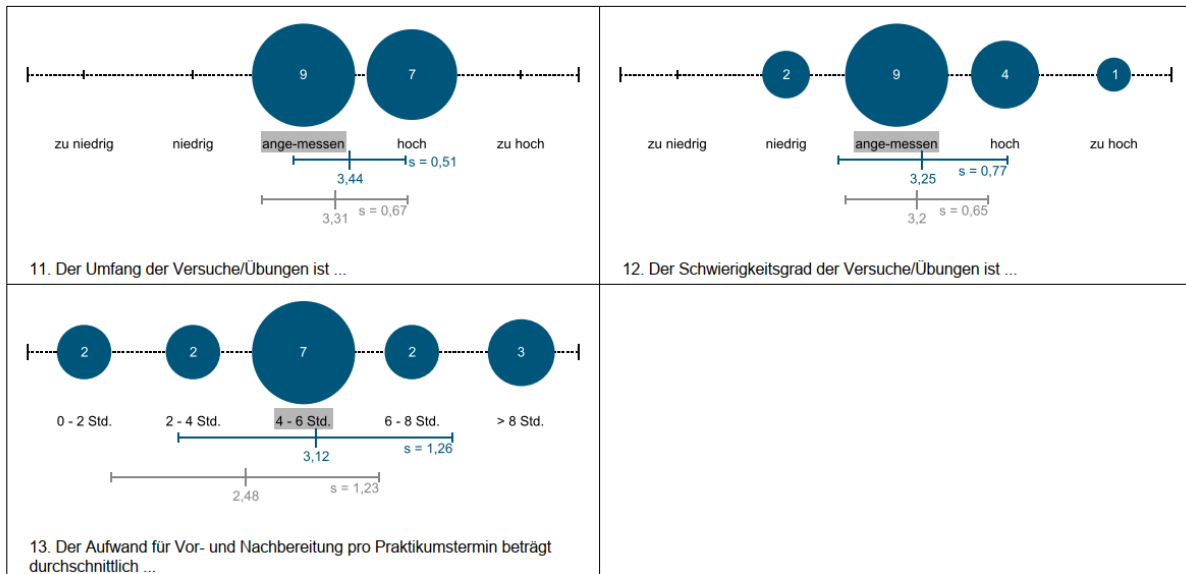


Eine Beispielklausur ist online auf der Web-Seite, die macht die Anforderungen transparenter.

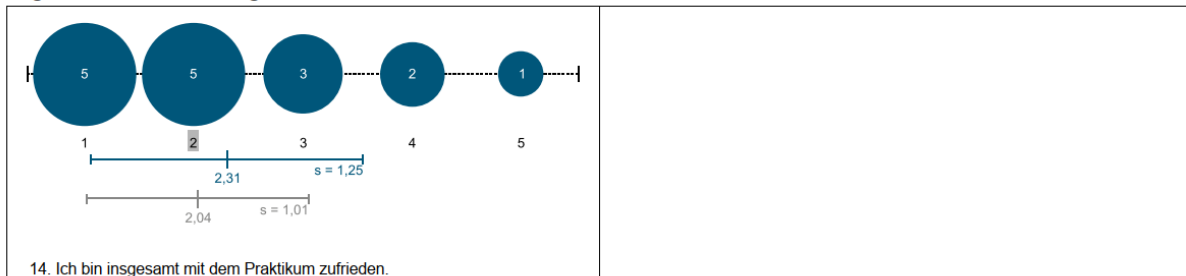
Bei dem Punkt „Umfang“ sehe ich das kritisch, da eine qualifizierte Person die gesamte Veranstaltung locker in 14 Stunden pro Woche erledigen können sollten.

Der Schwierigkeitsgrad der Aufgaben ist unter dem anderer IT-Studiengänge, bewusst, um möglichst vielen Studis den Einstieg zu ermöglichen (Vergleich von Online-Dokumenten).

WiSe 2025/26 (Fakultät Ingenieurwissenschaften und Informatik), Programmierung 1 (Praktikum) (Prof. Dr. Kleuker)



### Allgemeine Einschätzung



Anregungen und Kritik (IN DRUCKSCHRIFT) - wenn keine Anregungen oder Kritik, dieses Feld bitte freilassen.

- Das Beste an dem Praktikum ist die Freiheit und Flexibilität.
- Die Ausstattung der Computerräume ist sehr praktisch, es ist komfortabel, dass wir unsere eigenen Laptops zum Programmieren mit nur einem Kabel anschließen können und zwei Monitore (+Tastatur und Maus) zur Verfügung haben. Wenn die Räume frei sind, nutzen wir sie auch gerne in unseren Freistunden, weil das Arbeiten dort angenehmer ist als am "puren" Laptop.

Weil ich ein paar Vorkenntnisse habe, waren die Aufgaben für mich bisher eher leicht und zum größten Teil während der Praktikumszeiten zu schaffen, mit der steigenden Komplexität nehme ich an, dass der Zeitaufwand in den nächsten Wochen höher sein wird. Das AB 7 war das erste, an dem ich Zuhause noch länger als 2 Stunden weiter gearbeitet habe.

- Die Formatierung der Aufgabenstellungen ist sehr unübersichtlich.

Teilaufgaben im Fließtext, ohne Absätze dazwischen, tragen nicht dazu bei, die Aufgabenstellung sorgfältiger zu lesen, sondern verlangsamen das Arbeitstempo unnötig und bergen unnötige Fehlerquellen. (vgl. z.B. Aufgabenblatt 8 Aufgabe 28.)

Darum die Bitte: Einen Absatz zwischen den Teilaufgaben setzen, sodass man sie an Ihrer Struktur und nicht nur nach Suchen des Aufgabenbezeichners, ohne weitere Verwirrung und Ablenkung gezielt lesen kann.

Dies würde die Barrierefreiheit Ihrer Aufgabenstellungen erheblich verbessern.

Vielen Dank!

- Leider werden die Gruppen in den geplanten 45-Minuten-Testatterminen mittlerweile kaum noch mit den Vorstellungen ihrer Ergebnisse fertig, sodass diese im Praktikumstermin in Präsenz nachgeholt werden müssen. Dadurch fehlt wiederum Zeit, sich als Gruppe und ggf. mithilfe des Betreuers das aktuelle Aufgabenblatt zu bearbeiten. Dementsprechend wäre mehr Zeit zur Vorstellung der Arbeitsergebnisse oder alternativ asynchrone Bewertung (Abgabe → Prüfung → Korrektur) wünschenswert.

Über die Aufgabenformatierung werde ich nachdenken. Zum aktuellen Stand haben folgende Punkte geführt:

- Einige Studis hatten in Evaluationsgesprächen geäußert, dass Sie möglichst viele Informationen kompakt auf Aufgabenblättern haben wollten (wenige scrollen).
  - Das Studium bereitet auf die Praxis vor, da stehen oft relevante Informationen in Halbsätzen oder Fußnoten.
  - Eine Mischung aus Bildern und Text macht die Aufgaben schwerer für eine KI bearbeitbar; es finden KI-typische Auslassungen statt, wie falsch formatierte Säulen im Säulendiagramm.
  - KI kann aber helfen Aufgaben besser zu verstehen.
  - A28 ist nebenbei bewusst auf Lesbarkeit geschrieben worden; so werden exakt die Klausuraufgaben aussehen. Es gibt eine kompakte Aufgabendarstellung mit Nennung der Ausnahmefälle, danach folgen ein oder zwei erklärende Beispiele.
  - Bei Leseschwächen gibt es zum Glück den sehr sinnvollen Nachteilsausgleich.
- Ich werde natürlich trotzdem darüber ernsthaft nachdenken.

Bei den Testatterminen biete ich immer an, dass diese auch in der VL-Zeit nachgeholt werden können. Fragen per E-Mail sind auch immer erwünscht.