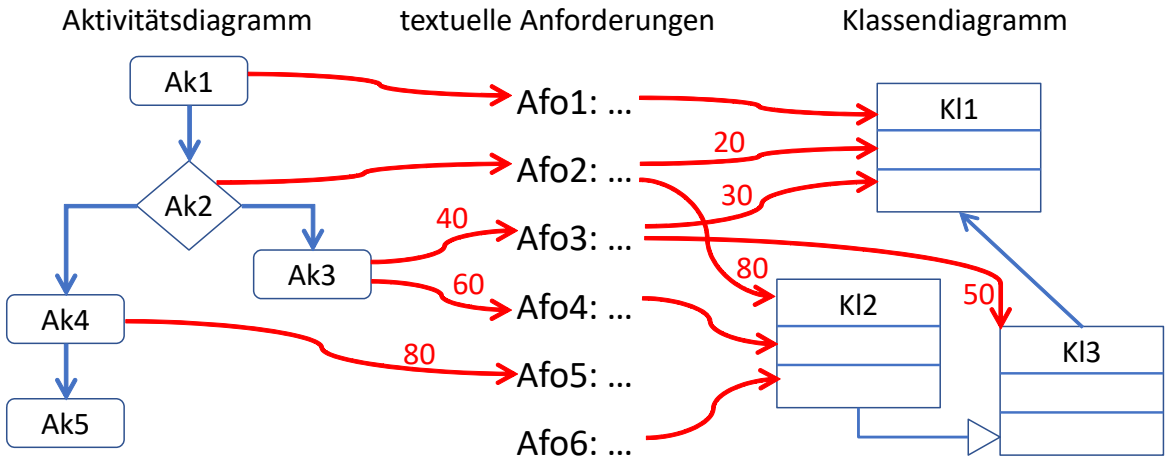
Prof. Dr. Stephan Kleuker
Fachhochschule Osnabrück
Fakultät Ing.-Wissenschaften und Informatik
- Software-Entwicklung -

Software Quality Management

Sommersemester 2023
3. Aufgabenblatt

Aufgabe 4

In der Vorlesung wurde gezeigt, dass es für die Nachverfolgbarkeit, also „was wurde mit was verknüpft/ wie umgesetzt“ von Entwicklungsentscheidungen wichtig ist, diese als Tracing-Informationen zu dokumentieren. In dieser Aufgabe wird dies mit einer Software umgesetzt. Eventuell hilfreich für die Begriffserklärung kann es sein, erst das folgende Diagramm zu verstehen (s. Vorlesung) und sich dann den Code mit den Ausgaben auf der folgenden Seite anzusehen.



Ziel der Aufgabe ist es, die Grundlage einer einfachen Tracing-Software in Go zu erstellen, die die Zuordnung sogenannter Artefakte ermöglicht. Artefakte können z. B. Aktionen eines Aktivitätsdiagramms (links), Anforderungen in Textform (mitte) und Klassen (rechts) sein. Die obige Abbildung zeigt eine Beispielzuordnung, bei der neben der Abhängigkeit auch ein prozentualer Anteil der Abdeckung angegeben wird. Nutzen Sie als Basistyp den vereinfachten Typ Artefakt aus dem Projekt von der Veranstaltungsseite. Dabei hat jedes Artefakt eine eindeutige id, einen fachlichen Inhalt und eine Sammlung von Gruppen, denen das Artefakt zugeordnet ist. Jede Gruppe wird durch einen string angegeben. Die Funktion New erzeugt ein Artefakt mit einer Gruppe.

```
type Artefakt struct {
    id      int
    inhalt  string
    gruppen set.Set /* Menge von strings, alternativ als Referenz */
}
```

Nutzen Sie weiterhin einen gegebenen Typ Verbindung, mit dem ein Artefakt einem anderen Artefakt zu einem bestimmten Prozentsatz zugeordnet werden kann.

```
type Verbindung struct {
    von     *artefakt.Artefakt
    nach    *artefakt.Artefakt
    prozent int
}
```

Implementieren Sie einen Typ Zuordnung, der eine Sammlung von Verbindungen enthält. Es soll immer nur eine Zuordnung zwischen zwei Gruppen möglich sein, z. B. von Aktionen zu Anforderungen.

Der Typ Zuordnung soll dabei u. a. folgende Funktionen realisieren.

```
liefert eine neue Zuordnung fuer Artefakte, die zur Gruppe „von“ gehoeren zu
Artefakten, die zur Gruppe „nach“ gehoeren (Ergebnis darf auch Zeiger sein)
    func New(von string, nach string) Zuordnung
```

```
ordnet dem Artefakt „von“ das Artefakt „nach“ zu „prozent“ Prozent zu, Ergebnis
zeigt an, ob Zuordnung erlaubt ist, also „von“ und „nach“ die richtigen Gruppen
haben und der Prozentwert sinnvoll ist; falls schon eine Zuordnung „von“ zu „nach“
existiert, wird diese ersetzt
        func (z *Zuordnung) Zuordnen(von *artefakt.Artefakt,
                    nach *artefakt.Artefakt, prozent int) bool

gibt alle Verbindungen aus, in denen dem Artefakt mit der Id „id“ ein Artefakt
zugeordnet wird
      func (z Zuordnung) Zugeordnet(id int) []*verbindung.Verbindung

gibt alle Verbindungen aus, in denen das Artefakt mit der Id „id“ einem Artefakt
zugeordnet wurde
      func (z Zuordnung) Abhaengig(id int) []*verbindung.Verbindung

gibt alle Artefakte aus, denen kein Artefakt zugeordnet wurde (Artefakte, die
nicht auf der „von“-Seite stehen und die zur Gruppe „von“ gehoeren)
      func (z Zuordnung) NichtZugeordnet() []*artefakt.Artefakt

gibt alle Artefakte aus, die zu keinem Artefakt zugeordnet wurden (Artefakte, die
nicht auf der „nach“-Seite stehen und die zur Gruppe „nach“ gehoeren)
      func (z Zuordnung) NichtAbhaengig() []*artefakt.Artefakt

gibt die Verbindungen aller Artefakte aus, denen keine Artefakte in der Summe von
100% zugeordnet wurden
      func (z Zuordnung) NichtVollstaendigZugeordnet()
                                          []*verbindung.Verbindung
```

Schreiben Sie dann einen Typen Zuordnungsliste, mit der eine beliebige Anzahl von Zuordnungen sequenziell verknüpft werden kann, dabei muss die Zielgruppe „nach“ einer Zuordnung immer die Ausgangsgruppe „von“ der nachfolgenden Zuordnung sein.
Die Klasse Zuordnungsliste soll wieder die letzten fünf genannten Funktionen des Typs Zuordnung realisieren, dabei muss Zugeordnet() nur für den Anfang und Abhaengig() nur für das Ende sinnvolle Ergebnisse liefern, die sich aber auf die gesamte Zuordnungsliste beziehen sollen. Die folgende im Projekt vorhandene Funktion erhält die obigen Beispieldaten als Eingabe.

```
func main() {
      ak1 := artefakt.New("Ak1", "Aktion") //(inhalt, erste Gruppe)
      ak2 := artefakt.New("Ak2", "Aktion")
      ak3 := artefakt.New("Ak3", "Aktion")
      ak4 := artefakt.New("Ak4", "Aktion")
      ak5 := artefakt.New("Ak5", "Aktion")

      afo1 := artefakt.New("Afo1", "Anforderung")
      afo2 := artefakt.New("Afo2", "Anforderung")
      afo3 := artefakt.New("Afo3", "Anforderung")
      afo4 := artefakt.New("Afo4", "Anforderung")
      afo5 := artefakt.New("Afo5", "Anforderung")
      afo6 := artefakt.New("Afo6", "Anforderung")

      kl1 := artefakt.New("Kl1", "Klasse")
      kl2 := artefakt.New("Kl2", "Klasse")
      kl3 := artefakt.New("Kl3", "Klasse")

      aa := zuordnung.New("Aktion", "Anforderung")
      aa.Zuordnen(ak1, afo1, 100)
```

```
        aa.Zuordnen(ak2, afo2, 100)
        aa.Zuordnen(ak3, afo3, 40)
        aa.Zuordnen(ak3, afo4, 60)
        aa.Zuordnen(ak4, afo5, 80)

        ak := zuordnung.New("Anforderung", "Klasse")
        ak.Zuordnen(afo1, kl1, 100)
        ak.Zuordnen(afo2, kl1, 20)
        ak.Zuordnen(afo2, kl2, 80)
        ak.Zuordnen(afo3, kl1, 30)
        ak.Zuordnen(afo3, kl3, 50)
        ak.Zuordnen(afo4, kl2, 100)
        ak.Zuordnen(afo6, kl2, 100)

        zli := zuordnungsliste.New()
        fmt.Println("aa hinzu: ", zli.Hinzu(aa)) // oder &aa
        fmt.Println("ak hinzu: ", zli.Hinzu(ak))
        fmt.Println("ak3 zugeordnet: ", zli.Zugeordnet(ak3.GetId()))
        fmt.Println("ak5 zugeordnet: ", zli.Zugeordnet(ak5.GetId()))
        fmt.Println("kl1 abhaengig: ", zli.Abhaengig(kl1.GetId()))
        fmt.Println("kl3 abhaengig: ", zli.Abhaengig(kl3.GetId()))
        fmt.Println("nicht zugeordnet: ", zli.NichtZugeordnet())
        fmt.Println("nicht abhaengig: ", zli.NichtAbhaengig())
        fmt.Println("nicht vollstaendig zugeordnet: ",
                zli.NichtVollstaendigZugeordnet())
    }
```

Die resultierende Ausgabe soll fachlich etwa wie folgt aussehen, andere Formatierungen und Reihenfolgen sind erlaubt. Weiterhin dürfen auch Pfade und weitere Methoden frei umbenannt werden.

```
aa hinzu: true
ak hinzu: true
ak3 zugeordnet: [[(3) Ak3 [Aktion] -> (8) Afo3 [Anforderung] 40%] [(3) Ak3
[Aktion] -> (9) Afo4 [Anforderung] 60%] [(8) Afo3 [Anforderung] -> (12) Kl1
[Klasse] 30%] [(8) Afo3 [Anforderung] -> (14) Kl3 [Klasse] 50%] [(9) Afo4
[Anforderung] -> (13) Kl2 [Klasse] 100%]]
ak5 zugeordnet: []
kl1 abhaengig: [[(6) Afo1 [Anforderung] -> (12) Kl1 [Klasse] 100%] [(7) Afo2
[Anforderung] -> (12) Kl1 [Klasse] 20%] [(8) Afo3 [Anforderung] -> (12) Kl1
[Klasse] 30%] [(1) Ak1 [Aktion] -> (6) Afo1 [Anforderung] 100%] [(2) Ak2 [Aktion]
-> (7) Afo2 [Anforderung] 100%] [(3) Ak3 [Aktion] -> (8) Afo3 [Anforderung] 40%]]
kl3 abhaengig: [[(8) Afo3 [Anforderung] -> (14) Kl3 [Klasse] 50%] [(3) Ak3
[Aktion] -> (8) Afo3 [Anforderung] 40%]]
nicht zugeordnet: [(5) Ak5 [Aktion] (10) Afo5 [Anforderung]]
nicht abhaengig: [(11) Afo6 [Anforderung]]
nicht vollstaendig zugeordnet: [[(4) Ak4 [Aktion] -> (10) Afo5 [Anforderung] 80%]
[(8) Afo3 [Anforderung] -> (12) Kl1 [Klasse] 30%] [(8) Afo3 [Anforderung] -> (14)
Kl3 [Klasse] 50%]]
```

Hinweis: Weder die genaue Architektur, noch die Nutzung von Referenzen, z. B. `Zuordnung` oder `*Zuordnung` in genannten Funktionen, noch Groß- und Kleinschreibung (Sichtbarkeiten), müssen ganz genau eingehalten werden. Sie dürfen auch über den Einbau eines Interfaces nachdenken. Wichtig ist hier nur, dass die angedeutete Funktionalität umgesetzt wird.