Fragen, Antworten, Kommentare zur aktuellen Vorlesung

Frage: Können oder sollen wir nicht Eclipse, Git und CodeTogether nutzen?

Dies sind sehr wichtige Werkzeuge im Laufe des Studiums, aber aus meiner Sicht frühestens ab dem zweiten Semester. Generell haben meine Erfahrungen in ersten Semestern gezeigt, dass jedwedes nützliche Werkzeug dazu führt, dass unsichere Studierende noch unsicherer werden, da sie zusätzlich das neue Tool verstehen müssen und so oft nicht zum Mehrwert kommen. Jede kleine zusätzliche Stufe führt zum Verlust von Personen am Anfang des Studiums.

Eclipse schauen wir kurz am Ende der Vorlesung an. BlueJ ist das einzige Werkzeug mit dem ein einfacher sauberer Einstig in die Objektorientierung funktioniert, was bei Studierenden zu besseren Lernerfolgen führt (Erfahrungen der Entwickelnden von BlueJ und auch meine Erfahrung). IntelliJ sieht besser aus, kann im Wesentlichen nicht mehr als Eclipse und erfüllt die Installationsanforderungen für die identische Form für die Hochschule und KleukersSEU nicht.

Git ermöglicht es systematisch verschiedene Versionen der gleichen Software zu verwalten und zugänglich zu machen. Das unterstützt einzelne Personen und Gruppenarbeiten mit gemeinsamem Git-Zugang. Git steht auf meiner Empfehlungsliste zum Selbststudium zwischen erstem und zweitem Semester.

CodeTogether ermöglicht, dass mehrere Personen gemeinsam in einem Editor Programmcode lesen und parallel bearbeiten können (live sharing). Das ist sehr cool und kann bei einer organisierten Zusammenarbeit sehr interessant sein. CodeTogether arbeitet dabei z. B. über Werkzeuggrenzen hinaus und eine Teilnahme im Browser kann sogar möglich sein. Ein Einführungstext steht in https://dzone.com/articles/remote-pair-programming-with-intellij-eclipse-and. Wieder ein tolles Programm, wenn alle Personen ein gewisses Programmier-Niveau erreicht haben. In den ersten beiden Semestern steht bewusst aber der Kampf „ein Mensch und eine Programmieraufgabe“ im Mittelpunkt und soll durch Gruppenarbeiten nur sinnvoll flankiert werden. Es ist auch möglich KleukerSEU um weitere PlugIns zu erweitern.

Frage: Kann es nicht mehr Unterstützung für MacOS geben?

Kurze Antwort: nein. Das Praktikum findet in der Hochschule unter Windows statt, als Bonus-Service wird die Umgebung auch für zu Hause nutzungsbereit angeboten. Später in Unternehmen geben die typischerweise das zu nutzende System vor. Jedes Betriebssystem mehr erhöht weiterhin den Arbeitsaufwand bei der Erstellung der SEU, da eventuelle Abhängigkeiten und Einschränkungen beachtet werden müssen.

MacOS ist eventuell für die reine Nutzung einfacher zu bedienen, für die Entwicklung in einer beliebigen Software mit Sicherheit nicht. Apple versucht diese Sprachenvielfalt zu verhindern, da Apple-Tools mit Apple-Sprachen genutzt werden sollen. Um andere Sprachen, wie Java zu nutzen, ist ein recht genaues Wissen über Installationspfade und Systemeinstellungen notwendig, die typischerweise über die Konsole erfolgen und zumindest nicht einfacher als in Windows sind. Starten programmierunerfahrene Studierende mit MacOS ist das wie beim Start eines Marathonlaufes mit zusammengeknoteten Schnürsenkeln loszulaufen. Das Ziel bleibt erreichbar, es muss aber einiges aufgeholt werden.

Kommentar: Ein gerne gemachter objektorientierter Fehler wird hier am Beispiel von 28 m) 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;
    }
  }
```

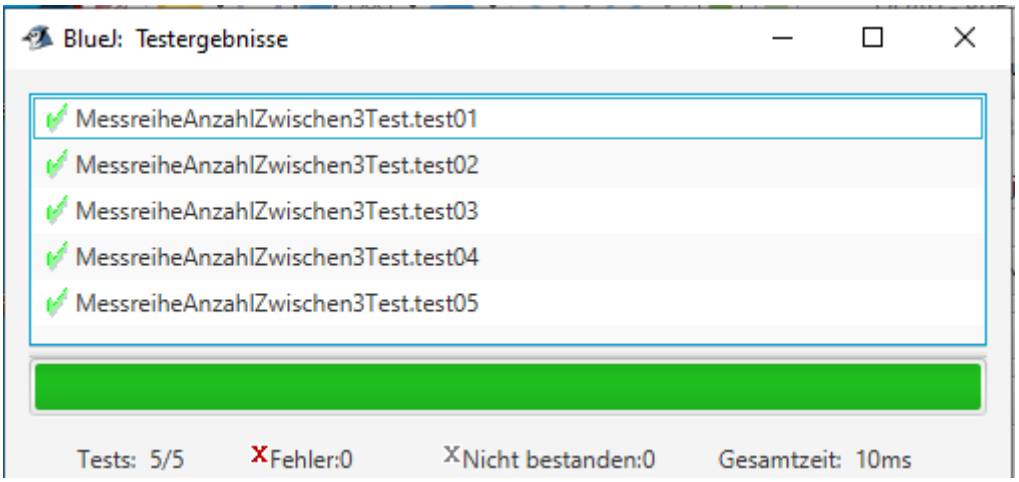
```
    return ergebnis;
  }

  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;
  }
```

Erinnerung: Tests helfen bei Entwicklung und ihre Nutzung ist in größeren Projekten ein Standard. Ob Tests vor oder nach der Entwicklung geschrieben werden, hängt wieder von Rahmenbedingungen ab. Generell muss man sich dazu merken, dass Tests notwendig (man muss sie machen) aber nicht hinreichend für eine gute Software-Qualität sind. Dies bedeutet, dass es trotz vollständig erfüllter Tests immer noch massive Fehler in den Programmen geben kann, siehe auch [Kle19], über die Bibliothek kostenlos herunterladbar. Dies bedeutet für Ihre Praktika, dass Ihre Programme trotz einer recht hohen Testanzahl noch falsch sein können und Sie Ihre Algorithmen weiterhin im Kopf überprüfen müssen. Wenn sowas nebenbei passiert, also falsche Programme mit laufenden Tests, schicken Sie mir solche Programme gerne zu, da ich dann zumindest die Testanzahl erhöhen kann (was nie hinreichend sein wird).

Ein sehr interessantes Beispiel ist diese vermeintliche Lösung zur Aufgabe 28 g)

```
public int anzahlZwischen3(int min, int max) {
  if (this.messwerte == null || this.messwerte.isEmpty()) {
    return 0;
  }
  int ergebnis = 0;
  for (int i : this.messwerte) {
    if (this.messwerte.get(i-1) <= max && this.messwerte.get(i-1) >= min){
      ergebnis = ergebnis + 1;
    }
  }
  return ergebnis;
}
```



Es sollte klar sein, dass hier die ForEach-Schleife nicht richtig verstanden und i nicht als Element, sondern als Index angesehen wurde. Da es mit get(i) eine Exception gibt, wurde mit i-1 experimentiert und siehe da, es klappt auch bei den korrigierten (siehe oben) Tests.

Den Fehler erkennt man mit Programmiererfahrung sehr schnell, aber Sie kommen immer wieder in Situationen, in denen Sie Anfänger auf einem Gebiet sind, so dass ein vergleichbarer Fehler dann auch möglich ist. Deshalb benötigt man z. B. beim Selbststudium immer einen erfahreneren Coach, der zumindest am Ende sich Ergebnisse anschaut.

[Kle19] S. Kleuker, Qualitätssicherung durch Softwaretests, 2. aktualisierte und erweiterte Auflage, Springer Vieweg, Wiesbaden, 2019