Musterlösung

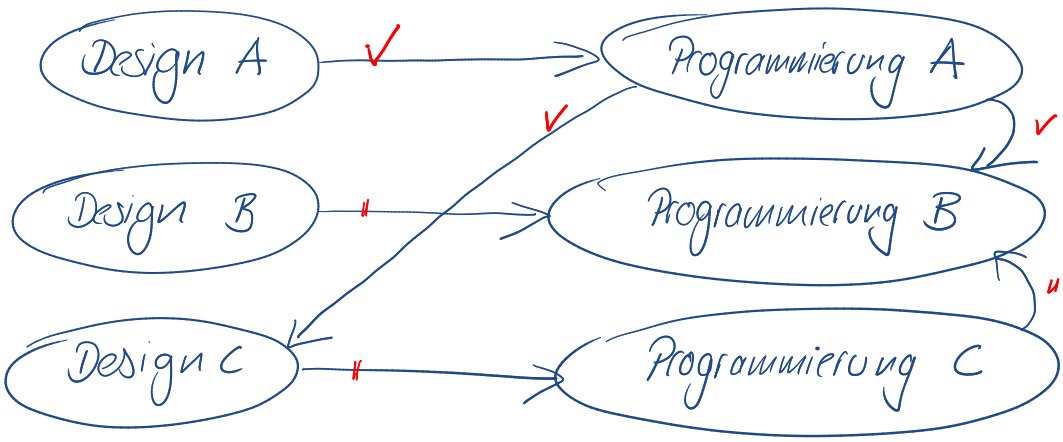
Aufgabe 1

a) Die Anforderungsanalyse ist die erste Phase eines Projektablaufs nach dem Wasserfallmodell und geht der genannten Phase unmittelbar voraus. Innerhalb dieser Phase werden die Ziele des Projekts, im Fall der Softwareentwicklung also die an die entstehende Software gestellten Anforderungen diskutiert und final festgelegt. Diese Anforderungen werden im Lastenheft festgehalten, welches als Resultat dieser Phase an die nun folgenden Phasen übergeben wird.

b) Es folgen noch die Implementierungsphase und eine Testphase. Die letzte Phase ist die Wartung der Software.

Aufgabe 2

a)



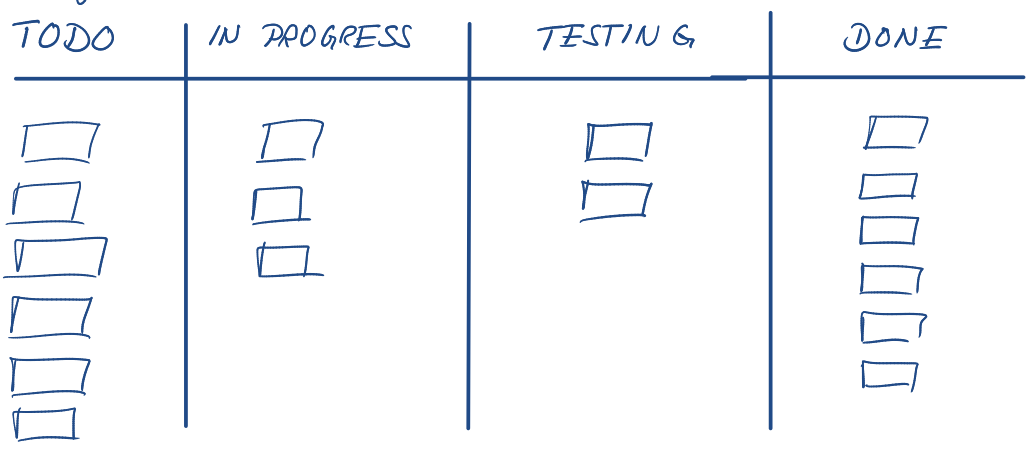
b) 1. Design A
2. Programmierung A
3. Design C
4. Programmierung C
5. Design B
6. Programmierung B

Anm: Bei stark vereinfachte a) nur maximal 1BE als folgerichtig bewertet.

c) Die Arbeitszeiten lassen sich nicht als Kantengewichtungen modellieren. Die Kanten aus dem in a) angegebenen Graph zeigen ja nur nötige Voraussetzungen für folgende Arbeiten. Gut sichtbar wird das bei Knoten „Programmierung B", von dem gar keine Kante weg führt, aber der offensichtlich mit einer Arbeitszeit verbunden ist.

Aufgabe 3

a) Das Kanbanboard ist ein wichtiges Hilfsmittel bei der agilen Softwareentwicklung. Es handelt sich dabei um eine physische oder virtuelle Tafel mit mehreren Spalten, in denen dynamisch Zettel mit den Aufgaben (user Stories) angebracht werden. Da jede Aufgabe mehrere Phasen durchläuft, wird nach Abarbeitung der entsprechenden Phase der Zettel in die nächste Phase verschoben. Diese Phasen werden auf dem Kanbanboard durch Spalten repräsentiert; in der Regel sind das:



Die Tafel ist für alle Mitarbeiter des Teams einsehbar und veränderbar. Wichtig ist sie aber auch in den gemeinsamen Treffen des Teams (z.B. Daily Scrum), bei denen die aktuellen Aufgaben und Probleme am Kanbanboard besprochen werden.

b) Kerngedanke und größter Vorteil der agilen Entwicklung ist die Möglichkeit auf Veränderungen jederzeit reagieren zu können. So kann beispielsweise eine Veränderung der Zielsetzung durch den Kunden innerhalb der Entwicklung aufgefangen werden. Beim Wasserfallmodell hingegen müssen alle Zielsetzungen sehr früh final festgelegt werden, da erst dann mit der eigentlichen Planung und Entwicklung begonnen wird.

Aufgabe 3

a) Bei dem abgebildeten Graphen handelt es sich um einen gerichteten Graph mit 7 Knoten und ganzzahlig gewichteten Kanten.

Die Klasse Graph nutzt das Prinzip der Adjazenzmatrix (s. Zeile 11) in einem zweidimensionalen Feld mit 20 × 20 ganzzahligen Werten. Jedem wird für jeden Knoten eine Bezeichnung als Zeichenkette gespeichert (siehe Zeile 10). Diese Art der Speicherung erlaubt ✓ die Speicherung des abgebildeten Graphen bzgl. der oben angegebenen Kriterien. Es können jedoch ✓ Probleme bei Nutzung der angegebenen Methoden auftreten: Die Methode „kanteAusgeben(...)" nutzt den Rückgabewert −1, falls zumindest einer der angegebenen Knoten nicht existiert, ansonsten wird die Gewichtung ausgegeben. Da aber zumindest eine Kante (C → F) mit Gewichtung −1 existiert, könnten Methodenrückgabewerte fehlinterpretiert werden. Ebenfalls muss ein explizites Wert (z.B. 0) in der Adjazenzmatrix als „Kante existiert nicht" reserviert werden, dies wird durch diese Implementierung nicht abgenommen. !

Zusammenfassend die Einschränkungen:

- maximal 20 Knoten ✓
- nur ganzzahlige ✓ Kantengewichtungen mit Problemen bei −1 und 0. ✓

b)

von \ nach	A	B	C	D	E	F	G
A		2	9	3			
B			8		4		
C						−1	2
D						2	
E			1				
F							4
G							

c)

```
besucht = boolean [ anzahl der knoten ]; // alle werte werden mit false instanzieert
void besuchen (startknoten)
    falls besucht [startknoten] == false :
        besucht [startknoten] = true;
        for alle direkten Folgeknoten i von startknoten :
            besuche (i);
        ende für alle
    ende falls
ende besuchen
boolean alleKnotenBesuchbar (startknoten)
    für alle Speicherplätze i in besucht :
        besucht [i] = false;
    ende for alle
    besuchen ( startknoten )
    für alle Speicherplätze i in besucht:
        falls besucht [i] == false :
            return false;
    ende falls,
    ende for alle
    return true
```