Übungsgruppe: Qianli Wang und Nazar Sopiha

Aufgabe 1-2:

a) Nein. Zum Beispiel, das Ziel “Qualität” braucht “Zeit” und/oder “Kosten”, genauso wie “Umfang”. Und große Umfang muss nicht hohe Qualität bedeuten. Alle vier Ziele sind im Prinzip miteinander verbunden. Wenn wir bei einem Ziel etwas gewinnen wollen, werden wir bestimmt auch etwas bei anderem Ziel verlieren.

b) **Niedrige Kosten: Health bracelet.** Wenn es kein Apple Watch ist, sondern etwas ganz billiges(20 Euro kostet), soll da auch ein billiges Software installiert sein, damit die Kosten von Software die Gesamtkosten nicht stark erhöhen.
Zeit: Coronavirus. Es ist in China noch erforderlich, einen Nachweis(durch ein App generiertes Security Code) vorzuliegen, wenn man ausgeht. Dann soll die Softwareentwicklung dringend in so kurzer Zeit wie möglich fertig sein, damit man ausgehen kann.
Qualität: Software für Flugzeuge. Es soll Fehlerfrei(Oder 99.999%) sein, weil die Kosten von Fehler unbestimmt und unglaublich groß sind.
Umfang: Wir sind mit diesem Punkt nicht ganz einverstanden. Wir glauben, dass Software die Technische Aufgabe erfüllen soll und (idealerweise) so gebaut werden, dass es auch leicht weiterentwickelt werden kann (**Wartbarkeit**). Vielleicht wird es in der Aufgabenstellung auch so gemeint, einfach nicht so klar formuliert.
Aus unserer Sicht soll das Ziel “Umfang” Vorrang haben, wenn es um ein großes, lang dauerndes Projekt geht, wo am Anfang wenige Funktionen implementiert sind, später aber ganz viel hinzugefügt werden kann. Beispiel - Google Maps, wo am Anfang einfach Maps waren, jetzt aber Route Builder, Timelines, Places To Go u.s.w.

c) Kosten und Zeitaufwand kann man eher leicht kontrollieren und sich dafür mit der Absicht entscheiden, wovon man mehr hat - Zeit oder Geld. Aber hohe Qualität und großer Umfang sind im Vergleich dazu schwieriger zu erreichen und zu messen, weil sie nicht so dringend relevant davon abhängig sind, wie viel Geld und Zeit man dafür engagiert. Trotzdem sind alle drei (vier) Ziele miteinander abhängig, deswegen kann man so eine Dreieckabhängigkeit bauen.

d) **Annahme:** Wir haben ein gutes Team, in dem man schon seit lange arbeitet, alle Kommunikationsprozesse gut eingestellt, Aufgabenverteilung gut organisiert.
Stichwort: Idealwelt
Alles hängt auch vom Projekt ab. Für (sehr) kleine Projekte ist es nicht so sinnvoll oder geeignet ein großes Team Einsatz gutzuschreiben. Und in diesem Fall arbeitet eine begrenzte Anzahl von Personen (z.B. 1 oder 2) und andere sind gar nicht produktiv. Die benötigte Zeit wird nicht entsprechend reduziert (5 Entwickler machen es nicht 5 mal schneller → **nicht proportional**).
Für große Projekte aber, wo man verschiedene Verantwortungssphäre bestimmen kann, ist es deutlich schneller ein großes leistungsfähiges Team im Einsatz zu bringen, denn in diesem Fall hat jeder Entwickler Verantwortung für eigene verteilte

Aufgaben, was den Entwicklungsprozess effizienter macht. Außerdem, für große Projekte kann man nicht mehr in allen Aspekten eine ausreichende Expertise haben, kann man nur in einer (zwei) Sphäre(n) ein Expert sein.
Vermutung: Zwei Entwickler brauchen insgesamt 5 bis 8 Monaten abhängig vom Projekt. Fünf → 2-4

Aufgabe 1-3:

a)

1. **Class** is an extensible program-code-template for creating objects,providing initial values for state and implementations of behavior. **z.B.** Klasse: Auto
2. **Vererbung**: Es gibt Elternklassen und ihre Kinderklassen können alle **Attributen** und **Methoden** erben und verwenden. In der Kinderklassen kann man auch neue Attribute und Methoden hinzufügen oder die von Elternklasse vererbte Methode überschreiben. **For example:** Elternklasse: Auto, Kinderklasse: BMW, BMW vererbt alles von Auto, hat auch eigene Eigenschaften.
3. **Objekt** ist ein **Exemplar** einer Klasse oder Konkretisierung von einer Klasse, die mit entsprechenden Werten initialisiert .
4. **Klasse** haben **Attribute -** das sind die Eigenschaften von jedem **Objekt.** Attribut von BMW Auto ist z.B. 0-100 km/h Beschleunigung
5. **Operation:** Operations just specify which functionality an object supports without any implementation. **For example:** a prototype in a C header.
6. **Methode:** Methods specify the concrete implementations an object supports. **For example:** an implemented C function.
7. **Exemplar:** One that serves as a model or example. **For example:** class fruit, the generated exemplar is apple.
8. **Bibliothek:** a collection of prewritten classes or coded templates that can be used for developing an application. **For example:** In C stdio.h for standard input and output functions
9. **Spezifikation:** The specification typically takes the form of a document that describes what the code is supposed to do, **for example** what sort of user interface options it should provide, what sort of output is expected for given inputs.
10. **Implementierung:** The actual embodiment in code that comprise the final product. **For example:** the functions that can be by an object invoked.
11. **Verifikation:** Verification is a way of preventing errors and making sure that we have gotten the right implementations. **For example:** Checksum
12. **Geheimnisprinzip:** The principle of segregation of the design decisions in a computer program that are most likely to change, thus protecting other parts of the program from extensive modification if the design decision is changed. **For example**: with “private” defined Code.

b) **Klasse** ist ein Template für Herstellung von beliebig vielen Objekten mit gleicher Struktur.

Menge von Objekten ist einfach eine Menge von schon hergestellten Objekten (nicht unbedingt mit gleicher Struktur), die auch möglicherweise aus unterschiedlichen Klassen stammen können.

c)

Aktivität: Management. Es gibt auch menschliche Probleme (Entwickler sind keine Computers), die von Managers gelöst werden sollen. Wie Zeitmanagement oder Aufgabenmanagement, Krankheit, Urlaub, Burnout, Änderungen(Laune,Leiter, Budget...) während des Entwicklungsprozessen u.s.w.

Aktivität: Inbetriebnahme. Aus unserer Sicht fehlt ganzes Testing-Verfahren