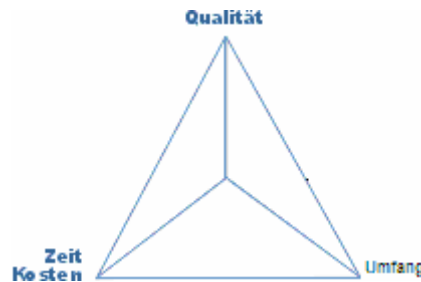


Dies ist keine „Musterlösung“, sondern eine gute von vielen möglichen Lösungen. Kommentare, die nicht Teil der Lösung sind, sind kursiv gesetzt.

### Aufgabe 1-2\*:

1. Die Ziele beeinflussen sich gegenseitig. Will man z.B. die Qualität optimieren, so braucht dies entweder mehr Zeit oder mehr Geld (weil man wahrscheinlich mehr Tester einstellen will). Man verdeutlicht dies gerne als Dreieck oder Tetraeder, wobei bei letzterem in der Spitze der Umfang ist. Ein Softwareprojekt ist ein Punkt innerhalb der aufgespannten Fläche und man muss sich als Projektverantwortlicher entscheiden, wo der Punkt liegen soll. Liegt er z.B. in der Nähe von (hohe) Qualität, dann ist der Punkt weit weg von den (niedrigen) Kosten und der (geringen) Zeit und dem (großen) Umfang. Bei Softwareprojekten ist wegen der dominierenden Gehaltskosten Zeit und Geld im wesentlichen äquivalent, so dass ein Dreieck Qualität - Kosten/Zeit - Umfang entsteht.



2. Qualität: Insulinpumpenregelung (bei Ausfall der Software hohe Gefahr)

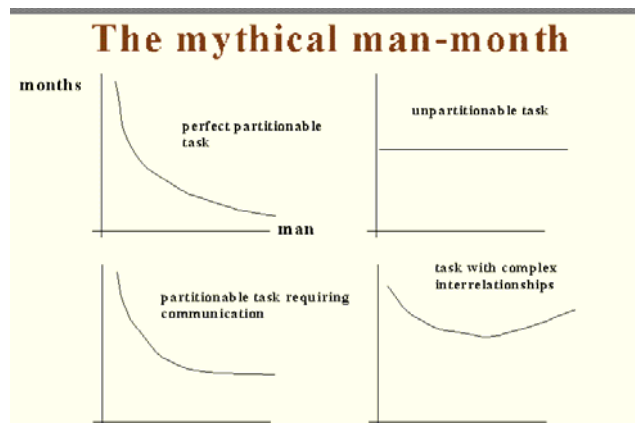
Kosten: (Schwierig) Forschungsprototypen (meist von Studierenden erstellt)

Es ist hier keine gute Lösung eine Massensoftware wie z.B. ein Betriebssystem oder ein Textverarbeitungsprogramm zu nennen, da diese sich gegen Konkurrenzprodukte am Markt durchsetzen müssen und man deshalb lieber etwas mehr Geld ausgibt um einen größeren Funktionsumfang der Software sicherzustellen als ein mittelmäßiges Produkt ohne Chancen herzustellen. Auch ist bei Massensoftware Qualität zumindest auf gleicher Wichtigkeitsstufe wie Kosten anzuordnen, da fehlende Qualität bei Auslieferung zu enormen Zusatzkosten führen kann.

Zeit: derzeit RFID-Integrationslösungen, d.h. heißes, nachgefragtes, neues Anwendungsgebiet (schnelle Marktreife ist derzeit wichtig)

Umfang: Textverarbeitung (Anwendungssoftware in Konkurrenz mit anderen, z.B. Word-Featuritis)

3. Zwei Leute brauchen nicht etwa nur ein halbes Jahr sondern länger. Das liegt an deren Koordinations- und Kommunikationsaufwand. 5 Leute brauchen entsprechend nicht 12 Monate / 5, sondern eher ein paar Monate mehr. Mehr als 3 Monate würde man allerdings nicht schätzen, wenn die Aufgabe auch nur ansatzweise unterteilt werden kann. 150 Leute werden mehrere Jahre brauchen, wenn man nicht 130 davon still sitzen lässt! Im folgenden Bild sieht man unten rechts den wahrscheinlichsten Fall.



### Aufgabe 1-3\*:

1. Es gibt natürlich viele, auch recht verschiedene Definitionen. Diese hier sind zu den Übungsblättern und der Vorlesung konsistent, nicht zwingend zu ALP.

- Klasse: Zusammenfassung von verschiedenartigen Elementen (members) mit gemeinsamem Gültigkeitsbereich unter einem Bezeichner. Elemente können Methoden oder Attribute sein.
- Vererbung: Eine besondere Beziehung zwischen zwei Klassen, in der
  - 1.) die erbende (Unter-)Klasse Methoden aus der vererbenden (Ober-)Klasse verwenden kann (Achtung: Sichtbarkeit! Bei private z.B. auch wieder nicht)
  - 2.) die Unterklasse die Attribute der Oberklasse beinhaltet (beinhaltet ist hier nicht klar definiert, da je nach Programmiersprache anders umgesetzt)
  - 3.) die Exemplare der Unterklasse typ-verträglich mit denen der Oberklasse sind, d.h. an allen Stellen wo nach einem Objekt der Oberklasse verlangt wird, kann der Programmierer auch ein Exemplar der Unterklasse übergeben.

- Man sagt: Die Unterklasse ist eine Erweiterung oder Spezialisierung der Oberklasse, die Oberklasse eine Generalisierung der Unterklasse.
- Attribut (engl. *field* häufig auch *Eigenschaft/property*): Die Aussage „Eine Klasse hat ein Attribut X vom Typ Y“ bedeutet, dass alle Objekte dieser Klasse eine (eigene) Variable mit Namen X und Typ Y besitzen. Hat die Klasse Auto z.B. ein Attribut mit Namen *lackFarbe* vom Typ *Color* und gibt es zwei Objekte dieser Klasse so kann eins dieser Objekte den Wert *rot* für dieses Attribut besitzen und das andere Objekt den Wert *blau*.
  - Man kann auch sagen, dass ein Attribut festlegt, welche möglichen Zustände die Objekte einer Klasse bzgl. dieses Attributs annehmen können.
- Methode = Operation: Eine Methode/Operation einer Klasse beschreibt eine Verfahrensvorschrift unter zu Hilfenahme der Attribute einer Klasse.
  - Beachte: Eine Methode/Operation wird in einer Klasse deklariert und wird von allen Objekten dieser Klasse gleichermaßen benutzt nur wird die Methode stets im Kontext eines Objektes aufgerufen und arbeitet dann auf den momentanen Attributwerten des Objekts.
  - Beachte 2: Klassenmethoden (und Attribute) haben wir in dieser ganzen Aufgabe nie betrachtet. Viele eurer Antworten hätten bei strenger Kontrolle nicht gestimmt, wenn man diese auch berücksichtigt hätte.
  - Der Begriff von Operation wird aber oft einschränkend für Infixnotationen der Art + \* - etc. benutzt. In UML hingegen gibt es nur Operationen, keine Methoden
- Exemplar = Objekt (= Instanz): Ein Objekt ist ein konkretes Exemplar einer Klasse zur Laufzeit des Programms.
  - Ein weiterer Begriff für dasselbe ist *Ausprägung*; das schöne an diesem ist, dass es auch ein Verb dazu gibt. Es ist nicht schön „Instanz“ (bedeutet im Deutschen was anderes, vgl. *Gerichtsinstanz*) oder „instanziieren“ zu sagen (das Wort gibt es im *Fremdwort-Deutschen* nicht)!
  - Zusatzwissen: In vielen Programmiersprachen (auch in Java) sind konkrete Klassen selbst auch Exemplare, nämlich Exemplare der Klasse *Klasse* (in Java heißt diese *Class*). In Java kann man die Klasse eines Objekts *a* durch *a.class* erhalten.
  - Achtung: Ein Objekt hat auch eine *Identität*, d.h. es kann zwei Objekte geben, die bzgl. allen Attribute gleich sind, aber trotzdem verschieden sind. Vergleiche dies mit primitiven Typen, bei denen dies nicht möglich ist.
- Bibliothek: Eine Bibliothek fasst eine Menge von Klassen und Methoden zwecks Wiederverwendung zusammen.
- Spezifikation einer Methode = Beschreibung ihres Kopfes („Syntax“) und ihres von außen sichtbaren Verhaltens („Semantik“) unter vollständiger Abstraktion von ihrer Implementierung (Rumpf). Eine Spezifikation beantwortet die Fragen „WAS leistet das System?“ oder „Was muss der Benutzer tun und was kann er erwarten?“ In der Softwaretechnik wird deutlich mehr spezifiziert als nur Methoden!
  - Achtung: Eine Spezifikation muss nicht formal sein.
- Implementierung beantwortet die Fragen „WIE erbringt das System seine Leistung?“ „Wie ist das System konstruiert?“
- Verifikation = Nachweis, dass die Implementierung der Spezifikation entspricht. Man sagt auch, dass die Implementierung die Spezifikation erfüllt.
- Geheimnisprinzip (information hiding principle):
  - Lokale Entwurfsentscheidungen innerhalb eines Moduls (z.B. einer Klasse) sollten von außen nicht sichtbar sein bzw. sollte sich ein Benutzer eines Moduls nur an die Spezifikation des Moduls halten und nicht auf interne Strukturen des Moduls zugreifen. Verletzt ein Programmierer nämlich ein solches Geheimnis und greift auf interne Methoden und Attribute zu, dann kann man das Modul nur noch viel schwerer ändern (weil sich der Programmierer ja darauf verlässt, dass das Modul nach einem gewissen Prinzip arbeitet). Noch schlimmer kann es sogar passieren, dass der Autor des Moduls eine Änderung am Modul durchführt, bei der sich das Verhalten einer internen Methode oder die Werte eines internen Attributs ändern, ohne zu wissen, dass der Benutzer des Moduls von diesen Werten abhängt. Fehler die so entstehen sind sehr schwer zu finden und häufig auch schwer zu beheben. Das Geheimnisprinzip wird in der Vorlesung noch wichtig!
  - Ein wichtiges Hilfsmittel um das Geheimnisprinzip umzusetzen ist die Sichtbarkeit (*private/protected/public*) von bestimmten Methoden/Attributen.
    - Achtung: Sichtbarkeit (*visibility*) ist nicht das gleiche wie Gültigkeitsbereich (*scope*).
  - Das wichtigere Hilfsmittel ist aber die Verwendung von Schnittstellen (*interfaces*).

2. Vor allem Anforderungserhebung und (Projekt-)Management. Sie waren bislang kein explizites Thema im Studium. Die Aktivitäten stehen auf z.B. Folie 21.

3. Eine Klasse beschreibt eine Art von Objekte (d.h. diese Objekte sind bezüglich der Attribute (nicht deren Werte) und der Methoden identisch) und ist keine Menge. Bei einer Menge müssen die Objekte nicht aus der gleichen Klasse kommen.