GAP: Notizblatt

Codierung von Zahlen
-Stellenwertsystem -> Dezimalsystem(Basis 10) und Binärsystem(Basis 2)
-Binärsystem für ganze Zahlen und Reele Zahlen
-N Bits = 2^N Zustände

Codierung von Texten
-im Computer im ASCII-Code (7 Bit pro Zeichen, für 125 Zeichen) oder Unicode gespeichert
-Unicode: ursprünglich 16-Bit, dann 21 (32)-Bit Kodierung und es werden nur 17 Ebenen genutzt

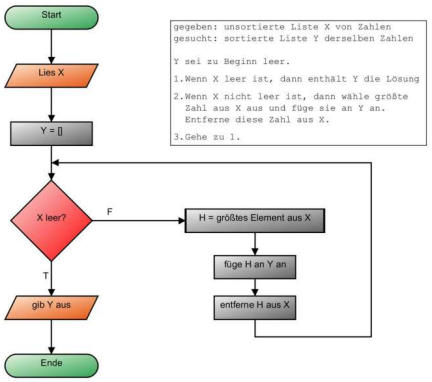
Codierung von Farben
-RGB-Farbmodell -> additive Farbmisch (rot, grün, blau) ->Farbe wird als Tripel aus jeweiligen Farbanteilen angegeben
-CMY(K)-Farbmodell -> subtraktive Farbmisch (cyan, magenta, yellow) ->Farbe entst durch Reflektion/Absorbtion an unters. Oberflächen
-YUV-Model -> Zerlegung der Farben in Helligkeitsanteil (Luminanz) und Farbanteil (Chrominanz)
-HSI-/HSL-Farbmodell -> Zerlegung der Farben in Farbton (Hue), Sättigung (Saturation) und Intensität (Intensity)

Codierung von Bildern
-Vektorgrafik -> Codierung von Linien, Polygonen und Kurven ->Ohne Qualitätsverlust beliebig skalierbar und Farbverläufe schwierig
-Rastergrafik -> Grafik wird in Matrix aus einzelnen Bildpunkten(Pixel) aufgerastert ->Pixel erhält Farb-/Helligkeitswert (Quantisierung)

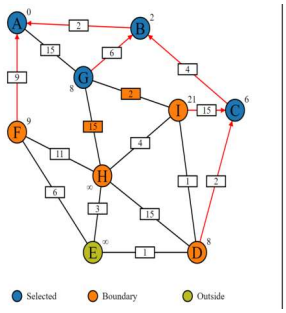
Codierung von Tönen
-Analog-Digital-Wandlung -> Abtastung des Signals (Sampling) ->Diskretisierung der Abtastwerte (Quantisierung)
->Kodierung der quantisierten Abtastwerte
-Tiefpassfilterung -> Frequenzanteile jenseits der kritischen Nyquist-Frequenz durch einen Tiefpassfilter entfernen
-Pulse-Code Modulation (PCM) -> Digitalisierung eines analogen Audiosignals

Verlustfreie Komprimierung
•Nachricht = übermittelte Zeichenkette, die meist nach bestimmten, vorgegebenen Regeln (Syntax) aufgebaut ist.
•durch Verarbeitung erhält die Nachricht Bedeutung (Semantik)
•durch die Verarbeitung der Nachricht ändert sich der Zustand des Empfängers der Nachricht (Pragmatik)
-Entropie -> Maß für den Informationsgehalt einer Nachricht
-Redundanz -> Anteile einer Nachricht, die keine zur Nachricht beitragende Information enthalten
-Redundanzfreiheit: ->PRO: -Datennehmen weniger Speicherplatz ein, -einfachere Änderung an Daten (müssen nur an einer Stelle gespeich.)
-Daten sind konsistenter und es gibt weniger Fehlerquellen und einfacher zu verstehen/verwalten
->CON: -Verwaltungsaufwand der Datenbank höher (Daten müssen an mehreren Stellen gespeich.)
-langsame Abfrage, weil mehr Tabellen durchs., -höherer Entwurfaufwand, weil Tabellenbz geplant muss
-Komprimierungsvarianten: Logische, Physikalische, Symmetrische, Asymmetrische, Nicht-adaptive, Adaptive, Semi-adaptive,
->Verlustfreie (Nach Kodierung+Dekodierung sind ursprüngl Daten unverändert u. verlustfrei rekonstruierbar)
->Verlustbehaftete (Beim Komprimieren gehen (unwichtige) Teile der ursprüngl Information verloren)
-Lauflängenkodierung: Bsp: AAABECCCFF ->2ABE3C2F
-Huffmancodierung: nach Häufigk. sortieren -> immer Elemente mit kleinst. Häufigkeit zsmfügen bis Wurzel -> Binärbaumkodierung -> Code
-LZW-Komprimierung: erzeuge aus zu kompr Zeichenketten Wörterbuch -> Daten mit Wörterbuch kodiert (komprimiert) -> Wörterbuch mit.
-JPEG-Kodierung: verlustbehaftete Komprimierung. -Ablauf: *Arbeitschritte/Speicherplatzersparnis(y,n)/Informationsverlust(y,n)*
(*komp. Bilder*) 1.Farbraumwechsel / NEIN / NEIN 2.Chroma-Subsampling / JA / JA 3.Diskrete Cosinus-Transformation / N / N
4.Quantisierung / J / J 5.Zig-Zag-Encoding / N / N 6.Lauflängencodierung / J / N 7.Modifiziert Huffman-C / J / N

Lösungen Modellieren
-Flussdiagramm: (Bsp.: Selection Sort)

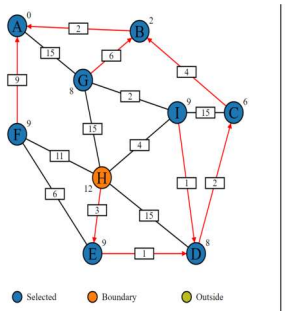


-Dijkstra: Dijkstras Shortest Path Algorit. berechnet **alle** kürzesten Wege von einem Knoten zu allen anderen Knoten in Graphen



Result

Node	Dist.	Pred.
A	0	-
B	2	A
C	6	B
G	8	B



Result

Node	Dist.	Pred.
A	0	-
B	2	A
G	6	B
J	8	B
C	8	C
H	9	A
F	9	D
I	9	D
E	9	D

Reguläre Ausdrücke

Regeln (cont): Zeichenklassen, z.B.

- **Ziffern, „Wortzeichen"(Alphanumerische Zeichen im engeren Sinn)**
- **\d: einzelne Ziffer**
- **\D: einzelnes Zeichen, das keine Ziffer ist**
- **\w: „Wortzeichen" (a-z,A-Z,0-9)**
- **\W: einzelnes Zeichen, das kein „Wortzeichen" ist**
- **\s: jede Form von Leerzeichen(space, tab, line feed)**
- **\S: einzelnes Zeichen, das kein Leerzeichen ist**
- **[a-b]: beliebiges Zeichen einer Klasse von a bis (einschließlich) b**
  - **Annahme: Zeichenklassen sind geordnet, so dass Ausdrücke der Form [a-b] sinnvoll interpretiert werden können.**

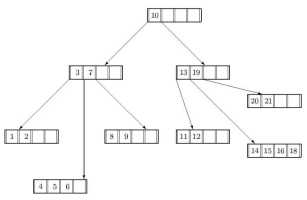
Datenstrukturen
-**Graphen**: -bestehen aus Knoten
-Kanten sind Verbindungen zw Knoten
- es gibt gerichteten o. ungerichteten Graphen
-Knoten sind benachbart, wenn sie Verbindung haben
-Eingangsgrad und Ausgangsgrad sind die Anzahl der Kanten, die von Knoten hingehen bzw. Weggehen
-wenn nicht gerichtet, dann ist der Grad die Anzahl der Kanten
-Ordnung des Graphen ist bestimmt, wie viele Knoten ich habe
-Größe des Graphen ist bestimmt, wie viele Kanten Ich habe
-Pfad in Graphen ist Abfolge von Knoten, die durch Kanten verbunden sind
-bei Schleife hat man Kante, von einem Knoten wieder zum gleichen Knoten hin
-Zyklus geht über mehrere Knoten, aber man kommt wieder zum Ausgangsknoten zurück

->Speicherung: - a) Adjazenzmatrix und b) Adjazenzliste
-Vorteile dieser: a) -einfache und schnelle Überprüfung, ob es eine Verbindung zw. 2 Knoten gibt
-geeignet für Graphen mit wenig Lücken (viele Verbindungen zw. Knoten)
b) -geeignet für Graphen mit vielen Lücken (wenig Verbindungen zw. Knoten)
-ermöglicht einfache und schnelle Implem. von Algorit. wie Tiefen-/Breitens.

-Begriffe:
•Graph: -Ansammlung von Knoten und Kanten ->Ordnung: Anzahl der Knoten ->Größe: Anzahl der Kanten
•Baum: -kreisfrei (keine geschlossenen Pfade) und zusammenhängend
•Gerichteter Graph/Baum: Kanten haben Richtung
•Gewurzelter Baum/Wurzelbaum: gerichteter Baum
•Binärbaum: Knoten haben maximal zwei direkte Nachkommen
•B-Baum / B+-Baum: komplexere Baumarten

-**B-Bäume**: Eigenschaften von B-Baum von Grad k:
1. Jeder Weg von Wurzel zu Blatt hat die gleiche Länge
2. Jeder Knoten außer der Wurzel hat mind. k und höchstens 2k Einträge
-> Wurzel hat höchsten 2k Einträge
-> Einträge werden in allen Knoten sortiert gehalten
3. Alle Knoten mit n Einträgen, außer den Blättern, haben n+1 Kinder
4. Seien S1, ..., Sn die Schlüssel eines Knotens mit n+1 Kindern
-> V0, ..., Vn seien die Verweise auf diese Kinder -> Dann gilt: .....

Bsp: B-Baum



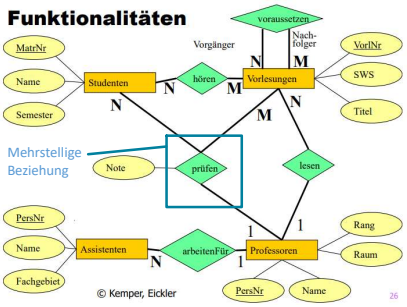
Regeln:

- **Jedes Zeichen ist ein Ausdruck, der das Zeichen repräsentiert.**
- **Der Punkt ist ein Spezialausdruck, der ein beliebiges einzelnes Zeichen repräsentiert**
- **Wenn A und B Ausdrücke sind, dann auch**
  - **AB: Sequenzen: Ausdruck A gefolgt von Ausdruck B**
  - **A|B: Alternativen: Ausdruck A oder Ausdruck B**
  - **(A): Gruppierung**

Regeln (cont):

- **Wenn A ein Ausdruck ist, dann auch**
  - **A*: Null oder beliebig viele Wiederholungen von A**
  - **A+: Eine oder mehr Wiederholungen von A**
  - **A?: Null oder ein Vorkommen von A**
  - **A{m}: genau m Wiederholungen von A**
  - **A{m,n}: zwischen m und n Wiederholungen von A**
  - **A{m, }: mindestens m Wiederholungen von A**

Problemlösung
-ER-Modell: besteht aus Entitäten ▭ | Beziehungen ♦ | Eigenschaften ⬭



-Prinzip „teile und herrsche": Zerlege Problem in überschaubare Teilprobleme
->kommt menschlicher Problemlösungsfähigkeit entgegen
->Hilft korrekte Lösungen zu erstellen
->Hilft robuste Lösungen zu erstellen (Austausch von Teilen problemlos mögl)
->Zerlegung Bsp: Mergesort, Mapreduce

Algorithmen
-Def: Algorithmus ist ein Verfahren mit einer präzisen (d.h. in einer eindeutigen Sprache abgefassten) endlichen Beschreibung unter Verwendung effektiver (im Sinne von tatsächlich ausführbarer) Verarbeitungsschritte.
-Eigens: •terminierend: Algorithmus heißt für eine Eingabe terminierend, wenn er stets nach endlich vielen Schritten endet.
•deterministisch: Algorithmus heißt für Eingabe deterministisch, wenn in der Auswahl der Verarbeitungsschritte keine Freiheit besteht.
•determiniert: Algorithmus heißt für eine Eingabe determiniert, wenn das Resultat des Algorithmus eindeutig bestimmt ist.
•sequentiell: Algorithmus heißt für eine Eingabe sequentiell, wenn die Verarbeitungsschritte stets hintereinander ausgeführt werden.
•parallel: Algorithmus heißt für Eingabe parallel, wenn gewisse Verarbeitungsschritte nebeneinander (gleichzeitig) ausgeführt werden.
-Sprachelemente: ->Datenspeicher und Zuweisungen (Variablen, Zuweisungen, Konstanten)
->Ausdrücke: -Boolesche Ausdrücke (A[j-1] > A[j])
-Arithmetische Ausdrücke (j := j-1)
->Sequenzierungsanweisungen (Blöcke, Funktionsaufrufe, Iterationsanweis.)
->Hilfskonstruckte (Kommentare)

-Sortieralgorithmen:

SELECTION SORT

Eingabe: unsortierte Liste X
Vorgehen:
- **Erzeuge leere Liste Y**
- **wähle kleinstes Element aus der Liste X aus**
- **Lösche es aus X**
- **Füge es ans Ende von Y an**
- **Wenn X leer ist: Gebe Y aus**

MERGESORT

Eingabe: unsortierte Liste X
Vorgehen
- **Rekursiv: zerlege Liste in Teillisten, sortiere Teillisten und mische sie dann rekursiv wieder**

INSERTION SORT

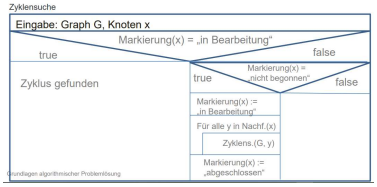
Eingabe: unsortierte Liste X
Vorgehen:
1. **Setze Markierung auf zweites Element der Liste**
2. **Vergleiche markiertes Element mit Vorgängerelement:**
   - Wenn kleiner: tausche die beiden
   - Wiederhole 2, bis kein Tausch mehr stattfindet
3. **Verschiebe Markierung um eins nach rechts**
   - Markierung nach Listenende: fertig
   - Sonst: gehe zu 2

-Struktogramm (Nassi-Shneiderman-Diagramme)

ZYKLENSUCHE IN GERICHTETEM GRAPHEN

```
1  prozedur Zyklensuche (Graph G, Knoten x)
2  falls Markierung(x) = „in Bearbeitung" dann Zyklus gefunden
3  sonst falls Markierung(x) = „noch nicht begonnen" dann
4        Markierung(x) := „in Bearbeitung";
5        für alle Nachfolgerknoten y von x tue Zyklensuche(y);
6        Markierung(x) := „abgeschlossen";
```

Zyklensuche



Adjazenzmatrix

	A	B	C	D	E	F	G
A		T	T				
B							
C		T				T	
D					T		
E				T			
F		T			T		
G							

Adjazenzlisten

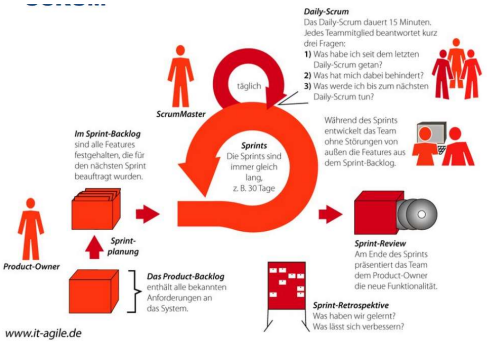
A	B, C
B	
C	B, F
D	E
E	D
F	B,E
G	

-**DNS**:Das Domain Name System (DNS) ist ein System zur Übersetzung von Domainnamen in numerische IP-Adressen, die von Computern und anderen Netzwerkgeräten verwendet werden, um miteinander zu kommunizieren. Das DNS besteht aus einem verteilten Netzwerk von Servern, die miteinander kommunizieren und Informationen über Domainnamen und zugehörige IP-Adressen bereitstellen. Es ermöglicht Benutzern, Websites und andere Netzwerkdienste über einfache Domainnamen zu finden und darauf zuzugreifen, anstatt sich die komplexen IP-Adressen merken zu müssen. DNS ist ein wichtiger Bestandteil der Infrastruktur des Internets und wird von nahezu allen Geräten und Anwendungen genutzt, die eine Internetverbindung benötigen.

Softwarequalität (Notizblatt Teil2)

•Chaos Report: IT-Projekte in 3 Gruppen eingeteilt
->Typ 1: Successful (*erfolgreich abgeschlossen*)
-On time, -On budget
-All features and functions as specified
->Typ 2: Challenged (*abgeschlossen, aber …*)
-Over time estimate, -Over budget
-Fewer functions and/or features
->Typ 3: Failed (*gescheitert*) ->Project was canceled
•Software-Lebenszyklus
-beginnt, wenn man über Software anfängt nachzudenken
-endet wenn man Software nicht mehr benutzt
-man unterteilt es in Phasen, welche Eingaben und Ausgaben hat
-man geht von einer abstrakten Idee zu einer festen Software

•**Scrum**:



•**SoftwareTests**:
-Ziele:-Funktionalität überprüfen
->Übereinstimmung mit den Anforderungen
-> geprüft ob Anforderungen erfüllt sind
->Für jede Anforderung gibt es mind. einen Test
-Validierungstest
->geprüft ob entwickelte System am Ende Aufgabe erfüllt
-Fehler aufdecken
->Verhindern von unerwünschtem Verhalten
->Auch "falsche" Eingaben möglich
•White box T: -Wissen über den inneren Aufbau einer Komponente
-> ich weiß was in Kasten drin ist
-Teilweise durch den Entwickler selbst
-> man testet mit dem Wissen, interne Komponenten
•Black box T: -man testet hier auch was in dem Kasten ist, man weiß aber nicht was drin ist, sondern nur was es tun soll
-Keine Kenntnisse über den inneren Aufbau der Komponenten, ich bin nicht Entwickler
-man weiß nur was Box tun soll an Hand von Spezifikation (und man testet das)
•Stufen des Testens
->Entwickler-Tests (development tests):
-Schon während der Softwareentwicklung
-im Besten Fall wird nach jeder Änderung getestet, ob alles noch lauft
->Release-Tests: -Vollständiges System wird getestet
-Optimal durch getrenntes Test-Team, also nicht von Entwicklerteam
->Nutzer-Tests (user tests): -Testen in eigentlicher Nutzerumgebung
-Durch Nutzer oder potentielle Nutzer getestet
•TDD:Test Driven Development
-Enge Verbindung von Entwickeln und Testen
-Zuerst Tests schreiben, danach Funktionalität entwickeln
-Hauptkriterium: Erfolgreiches Bestehen der Tests
-Inkrementelle Entwicklung -> Erst wenn alle Tests erfolgreich sind, wird das nächste Feature bearbeitet
->Prozess: -Neue Funktionalität beschreiben -> Sollte eher feingranular sein
-Tests entwickeln
-Ausführen des Tests -> Sollte beim ersten Ausführen fehlschlagen
-> Was wenn doch erfolgreich?
-Neue Funktionalität implementieren und Tests wieder ausführen
-Sobald alle Tests erfolgreich sind, nächste Funktionalität bearbeiten
->Vorteile-Test Coverage -> Jedes Code Fragment wird durch mind. einen Test überprüft
-Regression tests
-Simplified debugging -> Wenn ein Test fehlschlägt, sollte klar sein, wo das Problem ist
-> Neuen Code überprüfen
-System documentation -> Tests sind auch eine Art von Dokumentierung
->Regression Tests-Überprüfen, ob der neue Code nichts "kaputt" gemacht hat
-Im manuellen Testen sehr aufwendig - automatisiert aber einfach:
->Bei jeder Änderung alle Tests ausführen
-Alle Tests müssen erfolgreich sein, bevor eine Änderungen übernommen wird

BEISPIEL Turingmaschiene

Aufgabe: AAA+AA = AAAAA

Annahme: Lese-Schreibkopf steht zu Beginn beim ersten A

(z1,A) -> (z1,A,R) [wir befinden uns im ersten A-Block und bewegen uns nach rechts]

(z1,+) → (z1,A,R) [+ gefunden, wird durch A ersetzt, wir gehen weiter nach rechts]

(z1,_) → (z2,_,L)[wir haben das Ende der Folge erreicht und müssen jetzt nach links gehen, damit wir dort das letzte A löschen können]

(z2,A)→ (z2,_,H)[letztes A wird gelöscht (damit sind wieder so viele As wie am Anfang da, aber jetzt ohne + dazwischen), damit ist die Aufgabe erfüllt und wir bleiben stehen]

•**Wasserfallmodell**: 1.Feasibility (Anforderungen)
2.Requirement Analysis (Analyse)
3.System Architecture Design (Entwurf)
4.Implementation & Testing (Implem. & Testen)
5.Operation & Maintenance (Inbetriebnahme & Wartung)
-Softwareentwicklung als strikter "top-down" Prozess mit "stepwise infinement" und "milestones"
-es gibt Phasen, die entsprechende Milestones haben und dann wird Entwickelt
->Probleme: -strikt sequentieller Prozess
-spätes Testen und spätes Feedback an Kunden und Entwickler
-praktisch kein Risikomanagement für Kunden und Entwickler
->Alternative: -V-Modell, -Evolutionary Development, -Spiralen-Modell, -Unified Process, -Agile Ansätze, -etc.

•**Evolutionäre Softwareentwicklung**:
-geht davon aus, dass man Beschreibung hat, von dem Problem, was man lösen will u Idee was am Ende raus kommen soll
-man macht also Spezifikation und erstellt daraus erste Version
-wenn Version einigermaßen funktioniert, dann entwickelt man eigentliche Software
->Entwicklung in verschiedenen Phasen
->Man hat verschiedene Versionen (Initial, Intermediate, Final) , die Zwischendurch entstehen, anstatt Finale Version (wie in Wasserfallmodell)
-Versionen werden Kunden gezeigt um Feedback zu geben und zu bekommen
-auf Ende zu muss es validiert werden
-Ergebnis: Fertige Version
->PRO: -Spiegelt tatsächliche Bedürfnisse des Kunden wieder
-System kann an sich entwickelndes Problemverständnis des Kunden angepasst werden
-durch Zwischenversionen wird Kunden mehr Übersicht geboten und ihm wird genauer bewusst, was er möchte
->CON: -Prozess undurchsichtig (zu viele Pfeile, wann geh ich wo hin)
-Entstehender Code häufig unstrukturiert
-Gefahr, bei Zwischenversionen „hängen zu bleiben" und Endversion entsteht garnich

-man hat keine großen Zwischenversionen, die dem Kunden jeden Monat präsentiert werden
->man teilt es auf in kleine Unterabschnitte, "Sprints"
->Abschnitte sind immer gleich lang und Zeit ist festgelegt (wird nicht anhand von Milestones festgelegt)
-am Anfang der Periode, werden Probleme festgesetzt, die man lösen möchte (ToDoListe), wichtig sind und Zeitlich schaffbar sind
->innerhalb der Zeit werden nur diese Probleme bearbeitet
->nach der Zeit wird ausgewertet (was hat man geschafft und was nicht und warum)
-man hat tägliche kurze Meetings (was hat man am Vortag gemacht, Review, was macht man bis morgen)
->soll Mitglieder auf dem Laufenden halten, was im Team passiert
-wenn Kunde Sachen im Projekt ändern möchte (andere Anforderungen), dann wird dies innerhalb eines Sprints nicht geändert
->Forderungen des Kunden werden erst am Ende des Abschnittes erst angenommen und bearbeitet

->Product-Ownwer: -ist Interessenvertreter der Kunden im Team und hat klare Vorstellungen vom Produkt
-priorisiert die Aufgaben, die im Team erledigt werden müssen
-ist Ergebnisverantwortlich für die Kontinuierliche verbesserung und Wertmaximierung des Produkts
->Scrum-Master: -methodischer Experte
-coached kollegen und achtet auf Einhaltung des Scrum-Frameworks
-hält Störungen vom Team weg und räumt Hinternisse Weg, damit Entwickler ungestört arbeiten können
-sorgt dafür das richtige Personen eingesetzt werden
->Entwickler: -operative Fachexperten entscheiden, welche Aufgaben Sie erledigen, um das Produkt bzw. -bestandteile zu erstellen

•**User Story**:
-Eine User Story sollte enthalten:
-kurze, prägnante Beschreibung des Benutzerbedarfs oder -ziels
-Eine Erklärung, warum der Benutzer diese Funktion benötigt
-Eine Beschreibung der Benutzerrolle oder Personas, für die die Funktion gedacht ist
-Gegebenenfalls Akzeptanzkriterien, die Funktionalität spezifizieren u definieren, wann Funktion als abgeschlossen gilt
-Eigenschaften einer guten User Story: INVEST
-Independent: Funktionen können unabhängig von anderen User Stories implementiert werden
-Negotiable: Ausgestaltung kann verhandelt werden (kein „wie")
-Valuable: Sie nützt mindestens einem Stakeholder
-Estimable: Aufwand zur Umsetzung kann abgeschätzt werden, Problem und Lösung klar verstanden
-Small: Sie kann innerhalb eines Sprints umgesetzt werden
-Testable: Product Owner kann Kriterien festlegen, mit denen überprüft werden kann, ob Umsetzung abgeschlossen ist
-->Wenn eine User Story immer noch zu groß ist, um in einem Sprint bearbeitet zu werden, muss sie aufgeteilt werden
-Aufteilungsansätze:-Kann man die Story so aufteilen, dass
->Workflowbeginn und -ende in einer Story und die Mitte in einer anderen liegt
->Unterschiedliche Operationen in unterschiedlichen Stories beschrieben werden
->Teilmenge der Geschäftsregeln in der ersten Story, weitere in weiteren Stories umgesetzt werden
->Nach unterschiedlichen Daten aufgeteilt wird
->Entlang von Schnittstellen aufgeteilt wird
->Mit dem Teil mit dem größten Aufwand angefangen wird
->Erst eine einfache Lösung und dann komplexere entwickelt werden
->Erst mal Basislösung und dann performante Lösung entwickelt wird

Entwicklertests	Releasetests	Nutzertests
-Unit Tests -Komponenten Tests -System Tests -Teststrategien (Partition Tests, Guideline-based Tests) -Komponenten Testing -Interface-Testing -System-Testing -Use-Case-Testing	-Überprüfen einer Softwareversion, die auch außerhalb des Entwicklungsteams genutzt werden soll -Hauptzweck: Nachweis des Fit-for-Use -Meist BlackBox testing -Requirements based Testing -Performance Testing	-Durchgeführt von Nutzen bzw. Käufern ->Testen mit üblichem Workload -Trotz System und Release Tests noch nötig ->Alpha Tests: -Testen einer ersten (unfertige) Versionen der Software -Oft innerhalb der Entwicklungsumgebung ->Beta Tests: -Testen einer fertigen Version der Software -Nutzer können experimentieren und melden Probleme ->Akzeptanz Tests: -Der Käufer entscheidet, ob er das System als fertig ansieht. -Meist auf den Systemen des Käufers.

Zusammenfassung:
- Testen zeigt nur die Anwesenheit von Fehlern, nicht ihre Abwesenheit.
- Entwicklertests sollten durch die Entwickler selbst durchgeführt werden.
- Vor der Auslieferung sollte ein Test von einem anderen Team durchgeführt werden.
- Entwicklertests bestehen aus Unit Tests, die einzelne Komponenten testen, Komponenten Tests, die Gruppen von Komponenten testen, und System Tests, die das Zusammenspiel des ganzen Systems oder Teilen davon testen.
- Beim Testen sollte man explizit versuchen, Fehler zu finden.
- Tests sollten möglichst automatisiert werden, damit sie bei jeder Änderung einfach wiederholt werden können.
- TDD ist ein Verfahren, bei dem die Tests vor der Implementierung entstehen.
- Akzeptanztests überprüfen, ob ein System "gut genug" ist und produktiv eingesetzt werden kann.