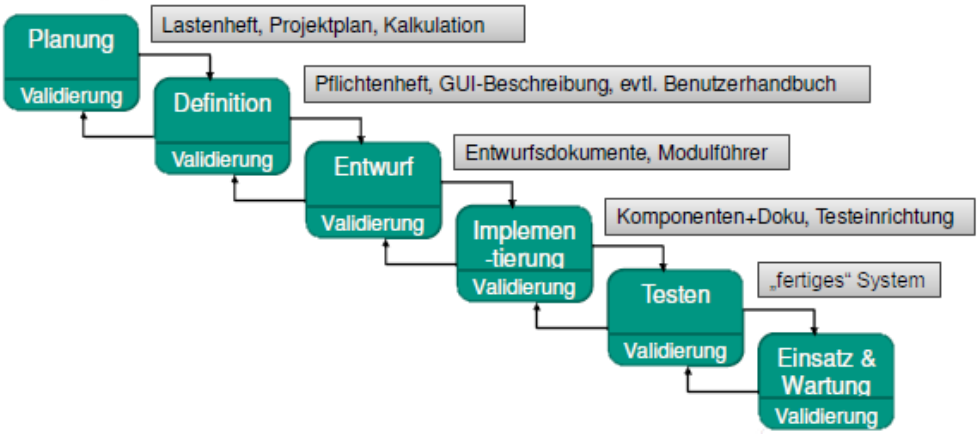
SWT – Zusammenfassung und Definitionen

Inhaltsverzeichnis

Wasserfallmodell:



- **Planungsphase**
  - Ergebnis
    - Durchführbarkeitsstudie
    - Lastenheft
    - Projektkalkulation
    - Projektplan
  - Ziele
    - Beschreibung des Systems in Worten des Kunden
    - Überprüfung der Durchführbarkeit

- Lastenheft
  - Zielbestimmung
  - Produkteinsatz
  - Funktionale Anforderungen
  - Produktdaten
  - Nichtfunktionale Anforderungen
  - Systemmodelle (Szenarien, Anwendungsfälle)
  - Glossar

- **Definitionsphase**
  - Ergebnis
    - Pflichtenheft / Produkt-Definition
    - Konzept Benutzungsoberfläche
    - Benutzerhandbuch
  - Ziele
    - Genaue Definition des zu erstellenden Systems
  - Modelle
    - Funktionales Modell
      - Szenarien
      - Anwendungsfall-Diagramm (Anforderungsermittlung / Zusammenhänge)
    - Objektmodell
      - Klassendiagramm (Beziehungen zwischen Klassen)
      - *Objektdiagramm (Interaktion zwischen Objekten)*
      - *Paket-Diagramm (Strukturierung)*
    - Dynamisches Modell
      - Sequenzdiagramm (Interaktion zwischen Objekten / Szenarien)
      - Zustandsdiagramm (Verhalten einer Klasse / Objektes)
      - Aktivitätsdiagramm (Abläufe)
  - Pflichtenheft (Bestehendes aus Lastenheft wird erweitert und Neues hinzugefügt)
    - *Zielbestimmung*
    - *Produkteinsatz*
    - Produktumgebung
    - *Funktionale Anforderungen*
    - *Produktdaten*
    - *Nichtfunktionale Anforderungen*
    - Globale Testfälle
    - Systemmodelle
    - *Glossar*

- **Entwurfsphase**
  - Ergebnis
    - Softwarearchitektur
    - Produkt-Entwurf
  - Ziele
    - Spezifikation und Gliederung des Systems in einer Bestandshierarchie
    - Antworten auf das WIE
      (z.B. Echtzeitbedingungen, Speicherung, Organisation, Installation, Ausnahmen, Rechte, Wiederverwendbarkeit, …)

        - Gute Abwägung der nichtfunktionalen Anforderungen
          (z.B. Laufzeit vs. Platzbedarf, Entwicklungszeit vs. Zuverlässigkeit, …)
        - Spezifikation jeder Systemkomponente
    - Entwurf
        - Modularer Entwurf
            - Schritte des Entwurfs:
                - **Modulführer**: Gliedert das System in Komponenten und Subsysteme mit genauer Beschreibung der Funktionen (Externer Entwurf).
                - **Modulschnittstellen**: Genaue Beschreibung der von jedem Modul zur Verfügung gestellten Elemente und verwendeten Ein-/Ausgabeformaten (Externer Entwurf).
                - **Benutztrelation:** Gliederung und Beschreibung der Interaktion der Module untereinander mit DAGs (Interner Entwurf).
                - **Feinentwurf:** Beschreibung der modul-internen Datenstrukturen und Algorithmen in möglichst Assemblerähnlichem Pseudocode (Interner Entwurf).
            - Anforderungen
                - Module werden ohne Kenntnis der späteren Nutzung oder Interaktion mit anderen Modulen entworfen, implementiert, getestet und überarbeitet.
                - Die Benutzung eines Moduls ist unabhängig vom Wissen über seinen inneren Aufbau.
        - Objekt-orientierter Entwurf
            - Folgt Prinzipien des modularen Entwurfs mit Klassen/Paketen statt Modulen und Paket-/Klassenführer statt Modulführer
            - Ermöglicht zusätzliche Beziehungen wie Mehrfach-Instanziierung, Vererbung, Polymorphie und Variantenbildung
    - Architekturstile und Entwurfsmuster
        - *S.u.*.
- **Implementierungsphase**
    - Ergebnis
        - Quellprogramme (Lauffähiger Code)
        - Objektprogramme
    - Ziele
        - Programmierung und Dokumentierung der Systemkomponenten anhand der Spezifikationen
- **Testphase**
    - Ergebnis
        - Test- und Verifikationsprotokolle
    - Ziele
        - Korrektheit des Systems und seiner Funktionalität überprüfen
        - Defekte jetzt und in Zukunft minimieren
    - Teststufen
        - Komponententest (s.u.)
        - Integrationstest (s.u.)
        - Systemtest

                - Funktionaler Systemtest (Überprüfung von Korrektheit und Vollständigkeit)
                - Nichtfunktionaler Systemtest (Überprüfung nichtfunktionaler Qualitätsmerkmale, z.B. Sicherheit, Benutzbarkeit, …)
            - Abnahmetest
- **Abnahme**, **Einsatz und Wartung**
    - Ergebnis
        - Installiertes Produkt
        - Gesamtdokumentation
        - Abnahme- und Einführungsprotokolle
    - Phasen
        - Abnahmephase
            - Übergabe von Produkt und Dokumentation an den Auftraggeber
            - Abnahme-, Belastungs- und Stresstests durchführen
            - Abnahmeprotokoll erstellen
        - Einführungsphase
            - Installation des Produkts
            - Schulung der Benutzer und des Betriebspersonals
            - Evtl. Umstellung / Übertragung von Datenbeständen
            - Inbetriebnahme des Produkts
                - Direkte Umstellung: *Unmittelbarer Übergang vom alten zum neuen System (risikoreich)*
                - Parallellauf: *Kurzzeitiger paralleler Betrieb beider Systeme zum Vergleich (Sicherheit, aber hohe Kosten)*
                - Versuchslauf: *Vereinzelter Übergang in Stufen mit Verwendung der Daten des alten Systems, welches weiterhin die aktuelle Verarbeitung durchführt*
            - Einführungsprotokoll erstellen
            - Anonymer Markt: Pilotinstallation / Betatest
        - Wartungsphase
            - Grund: Alterung der Software durch Umwelt und Anforderungen
            - Kategorien
                - Korrektive Tätigkeiten (Wartung)
                    - Stabilisierung/Korrektur
                    - Optimierung/Leistungsverbesserung
                - Progressive Tätigkeiten (Pflege)
                    - Anpassung/Änderung
                    - Erweiterung
            - Frage nach eigenständiger Wartung oder von Entwicklern aus
                - Vorteile
                    - Klare Zuordnung der Kosten
                    - Entlastung der Entwickler von Wartungsaufgaben
                    - Qualitativ besserer Abnahmetest
                    - Besserer Kundenservice durch Konzentration auf Wartung
                    - Effizientere Kommunikation zwischen Wartungsmitarbeitern

        - Höhere Produktivität durch Spezialisierung
    - Nachteile
        - Zwischen Entwicklung und Wartung kann Wissen verloren gehen
        - Koordinationsprobleme zwischen Entwicklung und Wartung
        - Entwickler tragen nicht die Konsequenzen ihrer Entwicklung
        - Wartungsmitarbeiter müssen sich einarbeiten
    - Mögliche Lösung: Rotation der Arbeiter

Wichtiges über Software:

Eigenschaften: immateriell, ohne Verschleiß, nicht durch phy. Gesetze begrenzt, einfacher und schneller zu verteilen und installieren als ein phys. Produkt, schwer zu „vermessen“

Zunahme in letzter Zeit an: Bedeutung/Komplexität/Anteil an mobilen Geräten/Vernetzung/Qualitätsanforderungen/Standardsoftware/Altlasten/“Außer-Haus“-Entwiclung

Anforderungen: Funktionstreue, Qualitätstreue, Termintreue, Kostentreue

Während der Entwicklung ändern sich: Produktanforderungen, Hardwarekomponenten, Portabilitätsanforderungen

Versionskontrolle:

Vergleich:	**SVN *(svn <cmd>)***	**Git *(git <cmd>)***
Vorteile:	- Internetfähig - versioniert das gesamte Projektdepot - atomares Einbuchen - optimistisches Ausbuchen	- Internetfähig - versioniert das gesamtes Projektdepot - Alles sind Objekte - Umgang mit Varianten - lokale Kopie des ganzen Depots
Nachteile:	- Keine eingebaute Semantik für Variantenbildung	- zusätzlicher Schritt zwischen lokalem und externen Depot
Ort des Depots:	Komplett auf SVN-Server	Server mit lokaler Kopie
Änderungen gespeichert als:	Deltas	Schnappschüsse
Befehle:		
Liste der Kommandos	*help <Kommando>*	*help <Kommando>*
Einpflegen ins Depot	*import <Pfad> <Depot>*	
Laden eines Teils des Depots	*checkout <Depot> <Pfad>*	*clone <Depot>*
Änderungen ins Depot übernehmen	*commit <Pfad>*	*commit <Pfad>*
Lokale Kopie aktualisieren	*update <Pfad>*	*fetch <Pfad>*
Neue Datei hinzufügen	*add <Datei>*	*add <Datei>*
Datei aus Depot löschen	*delete <Datei>*	*rm <Datei>*
Datei verschieben	*move <Datei> <Pfad>*	

Datei kopieren	*copy <Datei> <Pfad>*	
Verzeichnis anlegen	*mkdir <Pfad>*	
Neues Depot anlegen	*svnadmin create <Pfad>*	*init*
Logbuch anzeigen	*log [-v] <Datei>*	*log <Datei>*
Delta anzeigen	*diff –r<V1>:<V2> <Datei>*	*diff <V1> <V2>*
Änderung in ext. Depot übertragen		*push <Pfad>*
Änderungen von anderem Branch übertragen		*merge <Pfad>*

Maven Kommandos (*mvn <cmd>*):

compile – Erstellt Java-Bytecode-Dateien der Programm-Quelltexte

test – Erstellt Java-Bytecode-Dateien der Modultest-Quelltexte und deren Ergebnisse

package – Kompiliert, testet und verpackt das Projekt zu einem weiterverwendbaren Artefakt

install – Verpackt ein Artefakt und installiert es in die lokale Artefaktverwaltung

deploy – Schickt ein verpacktes Artefakt an eine öffentliche Artefaktverwaltung

Linguistische Analyse (Semantik statt Syntax):

Kürzel	Rolle	UML-Umsetzung
(AG) Agens	Der Handelnde	Klasse
(PAT) Patiens	Der Behandelte	Klassen, Methodenparameter
(ACT) Actus	Die Handlung	Methode
(INSTR) Instrumentum	Hilfsmittel	Klasse, Methodenparameter
(STAT) Status	Zustandsverben	Assoziation
(LOC) Locus	Orts-/Positionsangaben	Klasse
(OMN) Omnium	Das Ganze	Klassen, Komposition
(PARS) Pars	Ein Teil	
(DON) Donor	Der Gebende	Klassen, Multiassoziation, Methode zum Übergeben
(RECP) Recipiens	Der Empfänger	
(HAB) Habitum	Das Übergebene	

Architekturstile:

- **Schichtenarchitektur**
  - Eigenschaften
    - Aufteilung in Schichtenhierarchie
    - Jede Schicht greift nur auf Schichten darunter zu
  - Vorteile
    - Übersichtliche Strukturierung in Abstraktionsebenen oder VMs
    - Wiederverwendbarkeit, Änderbarkeit, Wartbarkeit, Portabilität und Testbarkeit ganzer Schichten wird unterstützt
  - Nachteile

        - Effizienzverlust bei intransparenter Schichtung
        - Chaosgefahr innerhalb einer Schicht
        - Klare Abgrenzung sind meist schwer möglich
    - Verwandte/Verwendete Entwurfsmuster
        - Fassade
    - Beispiele
        - Kommerzielle Webservices
        - Betriebssysteme
        - Protokolltürme
- **Klient/Dienstgeber**
    - Eigenschaften
        - Dienstgeber bieten Dienste für Klienten an
        - Anfragen werden vom Dienstgeber entgegengenommen, ausgeführt und zum Klienten zurückgegeben
    - Vorteile
        - Gebündelte Organisation im Dienstgeber
        - Erleichtert Änderung der Programme
    - Nachteile
        - Zentralisiert und damit anfällig für Fehler (z.B. Ausfall des Dienstgebers)
    - Verwandte/Verwendete Entwurfsmuster
        - Klient/Dienstgeber
    - Beispiele
        - Datenbanksysteme
        - FTP-Server
- **Partnernetze**
    - Eigenschaften
        - Verallgemeinerung des Klient/Dienstgeber-Stils
        - Alle Subsysteme sind gleichberechtigt (Rollensymmetrie)
        - Das Gesamtverhalten des Systems setzt sich aus der Interaktion der Partner zusammen (Selbstorganisation)
        - Jeder Partner kennt nur eine Untermenge der übrigen Partner = Nachbarschaft (Dezentralisierung)
        - Autonome Entscheidungen / Verhalten (Autonomie)
        - Partner können unzuverlässig / unerreichbar sein, weswegen Daten redundant und teils dupliziert existieren müssen (Verfügbarkeit)
    - Vorteile
        - Weniger ausfallgefährdet
        - Entlastet einzelne Partner
    - Nachteile
        - Höherer Aufwand bei Änderung der Programme
        - Viel Kommunikationsaufwand
    - Verwandte/Verwendete Entwurfsmuster
        - Klient/Dienstgeber
    - Beispiele
        - Torrents
        - TCP/IP
- **Datenablage**
    - Eigenschaften

        - Subsysteme modifizieren Daten einer zentralen Datenstruktur
        - Interaktion findet nur über Datenablage statt (lose Kopplung)
        - Mögliche Interaktionen: lokal/Fernzugriff, Warteschlangen/Parallelität
    - Vorteile
        - Zentrale Organisation der Zugriffe auf gemeinsame Daten
        - Jederzeit Einfluss auf Ausführung der Werkzeuge
        - Unabhängigkeit der verschiedenen Nutzer untereinander
    - Nachteile
        - Abhängigkeit von zentraler Stelle
        - Zentralisiert und damit anfällig für Fehler (z.B. Ausfall des Dienstgebers)
    - Beispiele
        - Subversion
- **Modell-Präsentation-Steuerung (MVC)**
    - Eigenschaften
        - Daten, Darstellung und Steuerung werden klar getrennt
        - Interaktion über wenige, wohldefinierte Schnittstellen
        - Steuerung (Controller) dient als Verarbeiter der Benutzerdaten und Kontaktstelle zwischen Modell (Model) und Präsentation (View)
    - Vorteile
        - Lose Kopplung zwischen Daten und Benutzerschnittstelle
        - Erleichtert Austauschen von Daten Modell oder Präsentation
    - Verwandte/Verwendete Entwurfsmuster
        - Beobachter
        - Fassade
    - Beispiele
        - Präsentationssoftware
        - Online-Formulare
- **Fließband**
    - Eigenschaften
        - Jede Stufe ist ein eigenständig ablaufender Prozess/Faden mit eigenem Befehlszähler
        - Daten fließen durch das Fließband und jede Stufe verarbeitet die Ergebnisse ihres Vorgängers
        - Puffer zwischen den Stufen dienen dem Geschwindigkeitsausgleich
        - Für die Verarbeitung von Datenströmen gedacht
    - Vorteile
        - Optimal für Parallelverarbeitung der verschiedenen Stufen
        - Kann auch leicht im Netzwerk verteilt werden
    - Nachteile
        - Engpässe einer Stufe kann das ganze Fließband aufhalten
        - Geschwindigkeit der Stufen sollte aufeinander angepasst sein
    - Beispiele
        - Unix Shell
        - Prozessor-Pipelines
        - Videobearbeitung
- **Rahmenarchitektur**
    - Eigenschaften

        - Definiert ein nahezu vollständiges Programm inklusive Anwendungslogik mit „Erweiterungspunkten“
        - Benutzer können durch Unterklassen dieser Lücken Methoden überschreiben oder abstrakte Methoden implementieren um die Funktionalität anzupassen oder zu erweitern (Plug-Ins)
        - „Don’t call us – we call you“ (Hollywood-Prinzip)
    - Vorteile
        - Weniger Entwicklungsaufwand durch Aufrüstung eines funktionierenden Systems
    - Nachteile
        - Plug-Ins können nur soweit definiert werden, wie die Rahmenarchitektur dies erlaubt
        - Abhängig von einer bereits fertig entwickelten, fehlerfrei funktionierenden Grundversion
    - Verwandte/Verwendete Entwurfsmuster
        - Strategie / Schablonenmethode
        - Fabrikmethode / Abstrakte Fabrik
    - Beispiele
        - Eclipse
        - Firefox
- **Dienstorientierte Architektur (SOA)**
    - Eigenschaften
        - Anwendungen werden aus (unabhängigen) Diensten zusammengestellt
        - Die Dienste sind dabei lose gekoppelt und über ein Dienstverzeichnis organisiert
        - Der Dienstnutzer sucht dabei im Dienstverzeichnis, bei dem die Dienste veröffentlicht sind, nach einem passenden Dienstanbieter
    - Beispiele
        - UDDI (Industrieübergreifender Verzeichnis-Standard für Webservices)

Entwurfsmuster:

Ziele:

- Wiederverwendung von Entwurfswissen
- Besser Kommunikation im Team
- Verständlichere Form von wesentlichen Konzepten
- Dokumentation und Förderung des Standes der Kunst
- Verbesserung von Code-Qualität und Code-Struktur

Liste der Muster:

- **Entkopplungsmuster**
  (Teilung eines Systems in mehrere Einheiten für bessere Unabhängigkeit mit Kommunikation über Schnittstelle)
    - Adapter *(Anpassung inkompatibler Schnittstellen)*
    - Beobachter (*Regelt Benachrichtigung abhängiger Objekte)*
    - Brücke (*Trennung von Abstraktion und Implementierung*)
    - Iterator (*Ermögliche sequentiellen Zugriff auf Elemente eines Objekts*)
    - Stellvertreter (*Kontrolliere/Überwache den Zugriff auf ein Objekt*)

    - Vermittler (*Kapselt das Zusammenspiel einer Menge von Objekten*)
- **Varianten-Muster**
  (Gemeinsamkeiten werden herausgezogen, um Wiederholungen zu vermeiden)
    - Abstrakte Fabrik (*Erzeugt Familien verwandter/abhängiger Objekte*)
    - Besucher (*Kapsele eine Operation als ein Objekt*)
    - Schablonenmethode (*Überschreibe bestimmte Schritte eines Algorithmus in den Unterklassen*)
    - Fabrikmethode (*Schnittstelle mit Operationen zum Erzeugen von Objekten*)
    - Kompositum (*Füge Objekte in Baumstrukturen zusammen*)
    - Strategie (*Definiere austauschbare Familien von Algorithmen*)
    - Dekorierer (*Fügt dynamisch Funktionalität zu bestehendem Objekt hinzu*)
- **Zustandshandhabungs-Muster**
  (Bearbeitung des Zustandes von Objekten unabhängig vom Zweck)
    - Einzelstück (*Stelle sicher, dass nur ein Exemplar einer Klasse existiert*)
    - Fliegengewicht (*Speichere große Mengen von ähnlichen Elementen*)
    - Memento (*Erfasse und externalisieren den inneren Zustand eines Objektes*)
    - Prototyp (*Erzeuge ein typisches Element eines Objekttyps und erlaube Duplikation*)
    - Zustand (*Definiere Verhalten eines Objektes abhängig seines Zustandes*)
- **Steuerungs-Muster**
  (Steuerung des Kontrollflusses)
    - Befehl (*Kapsele einen Befehl als ein Objekt*)
    - Master/Worker (*Verteile Arbeit auf identische Arbeiter*)
- **Virtuelle Maschinen**
  (Enthalten Daten und führen ein gegebenes Programm mit diesen Daten selbstständig aus)
- **Bequemlichkeitsmuster**
  (Sparen Schreib- und Denkarbeit)
    - Bequemlichkeits-Klasse (*Kapsele Methodenparameter in eigener Klasse*)
    - Bequemlichkeits-Methode (*Überlade Methoden mit häufig benutzen Parametern*)
    - Fassade (*Biete einheitliche Schnittstelle für ein ganzes Subsystem*)
    - Null-Objekt (*Inaktiver Stellvertreter*)

Parallelisierung:

Warum:

- Mehr Leistung und mehr Ressourcen benötigt
- Übersteigen der technischen Grenzen für Ein-Prozessorsysteme
- Möglichkeit der Verteilung auf mehrere Rechner/Systeme

Folgen

- Hauptproduktlinie der Prozessor-Hersteller sind Multikern-Chips
- Leistungssteigerung im Alltag durch Parallelisierung
- Parallelismus wird zum Normalfall
- Industrieweite Umstellung auf Parallelverarbeitung

Java

- Möglichkeiten zur Parallelisierung
    - Implementieren der Schnittstelle Runnable

        - Vorteile
            - Die gekapselte Aufgabe ist mit weniger Overhead in einem sequenziellen Kontext verfügbar
            - Kann im Netzwerk versendet werden da serialisierbar
    - Anlegen einer Subklasse von Thread
- Koordination
    - Wechselseitiger Ausschluss
        - Monitore
        - Synchronized-Block / Synchronized-Methode
        - Concurrent-Datenstrukturen
            - Semaphore (Verwaltet „Genehmigungen“)
            - CyclicBarrier (Wartet auf n Fäden)
    - Warten auf Ereignisse
        - Wait() / Notify()
        - Join()
    - Unterbrechungen
        - InterruptedException / interrupt()

Verfahren zur Parallelisierung

- Gebietszerlegung (Matrix-Vektor-Multiplikation)
- Master/Worker (Entwurfsmuster)
- Erzeuger/Verbraucher (mit Puffern)
- Fließband (Entwurfsmuster)
- Teile-und-Herrsche (Algorithmik)
- Map/Reduce

Bewertung

- Sequentieller Zeitaufwand: σ
- Parallelisierbarer Zeitaufwand: π
- Zeitaufwand: $T(k) = \sigma + \pi / k$ (Zeit für Ausführung mit k Fäden)
- Beschleunigung: $S(k) = T(1) / T(k)$ (Beschleunigung der Zeit mit k statt 1 Faden)
- Effizienz: $E(k) = S(k) / k$ (Anteil an Ausführungszeit für jeden Faden)
- Amdahlsches Gesetzt: $S(k) \leq (\sigma + \pi) / \sigma$

Definitionen der Testphase:

Fehlerarten:

- Versagen / Ausfall: Abweichung des Verhaltens der Software von der Spezifikation
- Defekt: Mangel in einem Softwareprodukt, der zu einem Versagen führen kann.
- Irrtum / Herstellungsfehler: Menschliche Aktion, die einen Defekt verursacht.

Testhelferarten:

- Stummel: Rudimentär implementierter Teil der Software als Platzhalter für noch nicht umgesetzt Funktionalität
- Attrappe: Simuliert die Implementierung zu Testzwecken
- Nachahmung: Attrappe mit zusätzlicher Funktionalität, wie bspw. das Einstellen der Reaktion der Nachahmung auf bestimmte Eingaben oder das Überprüfen des Verhaltens des „Klienten“

Fehlerklassen:

- Anforderungsfehler (Defekt im Pflichtenheft)
- Entwurfsfehler (Defekt in der Spezifikation)

- Implementierungsfehler (Defekt im Programm)

Testverfahrenskomponenten:

- Softwaretest / Test: Führt eine einzelne Softwarekomponente unter bekannten Bedingungen aus und überprüft ihr Verhalten (Ausgaben/Reaktionen).
- Testling / Testobjekt: Die zu überprüfende SW-Komponente
- Testfall: Besteht aus einem Satz von Daten für die Ausführung eines Teils oder des ganzen Testlings.
- Testtreiber / Testrahmen: Versorgt Testlinge mit Testfällen und stößt die Ausführung der Testlinge an

Testphasen (s.u.):

- Komponententest: Überprüft die Funktion eines Einzelmoduls durch Beobachtung der Verarbeitung von Testdateien
- Integrationstest: Überprüft schrittweise das fehlerfreie Zusammenwirken von bereits einzeln getesteten Systemkomponenten.
- Systemtest: Abschließender Test des Auftragnehmers in realer Umgebung ohne Kunden.
- Abnahmetest: Abschließender Test in realer Umgebung unter Beobachtung, Mitwirkung und/oder Federführung des Kunden.

SW-Inspektionsarten:

- Durchsicht: Der Entwickler führt Kollegen durch einen Teil des Codes / Entwurfs. Es werden Fragen und Anmerkungen zu Stil, Standards, Defekten, … gestellt.
- Überprüfung: Formalisierter Prozess zur Überprüfung durch einen „externen“ Gutachter
- Inspektion: Überprüfung anhand von Prüflisten und Lesetechniken
  („Die Inspektion ist eine formale Qualitätssicherungstechnik, bei der Anforderungen, Entwurf oder Code eingehend durch eine vom Autor verschiedene Person oder eine Gruppe von Personen begutachtet werden. Zweck ist das Finden von Fehlern, Verstößen gegen Entwicklungsstandards und anderen Problemen.“)

Lesetechniken:

- Ad-Hoc: Ohne Vorhaben
- Prüflisten: Liste mit Fragen, die (dokumentenabhängig) abgehakt werden
- Perspektiven/Szenarien: Erweitern Prüflisten um bestimmte Sichtweisen / Arten von Defekten

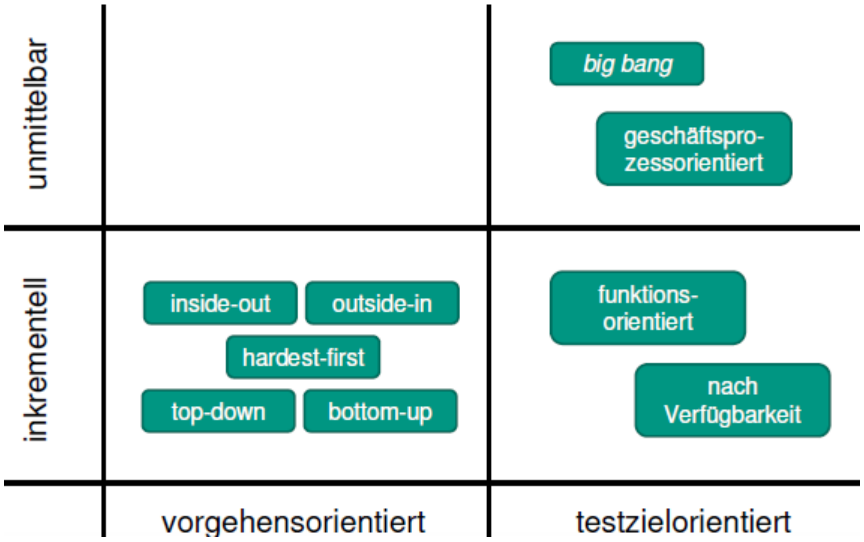
Komponententests:

- **Dynamische Verfahren** *(Testen Ausführung)*
  - Strukturtests *(z.B. Kontrollflusstests)*
    - Anweisungsüberdeckung
      (Alle Knoten im KFG werden durchlaufen)
    - Zweigüberdeckung
      (Alle Kanten im KFG werden durchlaufen)
    - Pfadüberdeckung
      (Alle Pfade im KFG werden durchlaufen)
    - Bedingungsüberdeckung
      - Einfache BÜ
        (Alle atomaren Bedingungen mit 1/0 belegen)
      - Mehrfache BÜ
        (Alle Bedingungskombinationen belegen)

                - Minimal-mehrfache BÜ
                  (Alle Bedingungsblöcke betreten)
    - Funktionale Tests
        - Ziel
            - Testen der spezifizierten Funktionalität durch aus der Spezifikation abgeleitete Testfälle
        - Verfahren
            - Funktionale Äquivalenzklassenbildung
              (Für Repräsentantenwerte verhält sich das Programm gleich wie für alle anderen Werte dieser ÄK)
            - Grenzwertanalyse
              (Testen der mit Randelementen der ÄK)
            - Zufallstest
              (Testeingaben zufällig auswählen, vermeiden von „Betriebsblindheit“ beim Testen)
            - Benutzung von Testhelfern
              (Minimieren Abhängigkeiten von anderen Komponenten beim Testen)
            - Test von Zustandsautomaten
    - Leistungstests *(Nichtfunktionale Anforderungen)*
        - Testen nahe der gegebenen Grenzen
        - Testen beim Überschreiten der Grenzen
            - Leistungsverhalten?
            - Rückkehr zu funktionierendem Verhalten?
- **Statische Verfahren** *(Prüfen Quellcode)*
    - Manuelle Prüfmethoden (*z.B. Review, Walkthrough, …)*
        - Naive Methoden *(unzureichend, mühsam)*
            - Ausgabeanweisungen
            - Interaktiver Debugger
            - Test-Skripte
        - Software-Inspektionen
            - Vorteile
                - Anwendbar auf alle Softwaredokumente
                - Jederzeit durchführbar
                - Sehr effektiv
                - Inspektoren bringen Erfahrung und haben Abstand
            - Nachteile
                - Aufwendig von Hand
                - Hoher Zeitaufwand
                - Statisch
            - Phasen
                - Vorbereitung
                  (Festlegen der Rollen und Techniken, Vorbereitung des Ablaufs und der Dokumente)
                - Individuelle Fehlersuche
                  (Dokumentierung im Alleingang)
                - Gruppensitzung
                  (Sammeln, Besprechen und Vorschläge zur Verbesserung)

        - Nachbereitung
          (Editor erhält fertige Liste, klassifiziert Defekte und überprüft Bearbeitung)
        - Prozessverbesserung
          (Anpassung der Inspektion)
    - Fortgeschrittene Methoden
        - Zusicherungen
        - Automatisch ablaufende Testfälle
    - Prüfprogramme
        - Überprüfen…
            - Warnungen und Defekte
            - Programmierstil *(z.B. Checkstyle)*
            - Fehlermuster *(z.B. FindBugs)*

Integrationstests:



- **Typen**
    - Unmittelbar (U)
        - Direkter Test am gesamten System (nur für kleine Systeme geeignet)
        - Vorteile
            - Keine Testtreiber oder Platzhalter benötigt
        - Nachteile
            - Alle Systemkomponenten müssen fertig sein
            - Defekt schwer zu lokalisieren
            - Testüberdeckung schwierig sicherzustellen
    - Inkrementell (I)
        - Teste immer größere Mengen von Komponenten
        - Vorteile
            - Testen fertiger Komponenten und Fertigstellung unfertiger Komponenten parallelisierbar
            - Testfälle leichter konstruierbar
            - Testüberdeckung prüfbar
        - Nachteile

        - Evtl. viele Testtreiber und/oder Testhelfer notwendig
    - Vorgehensorientiert (V)
        - Integrationsreihenfolge leitet sich aus der Systemarchitektur ab
    - Testzielorientiert (T)
        - Testfälle werden ausgehend von den Testzielen erstellt
        - Jeweils benötigte Komponenten werden „zusammenmontiert“
- **Strategien**
    - Big Bang (UT)
        - Alle Komponenten werden gleichzeitig integriert
        - Integrationstest kaum systematisierbar
        - „Nichts geht bis alles geht“
    - Geschäftsprozessorientiert (UT)
        - Alle vom Geschäftsprozess / Anwendungsfall betroffenen Komponenten werden zusammen integriert
    - Funktionsorientiert (IT)
        - Spezifikation funktionaler Testfälle
        - Orientiert an funktionalen Anforderungen
        - Schrittweise Integration und Test der Komponenten, die durch die Testfälle betroffen sind
    - Nach Verfügbarkeit (IT)
        - Integration von Komponenten sofort nach Abschluss ihrer Überprüfung
        - Schrittweise Integration und Test
        - Reihenfolge durch Implementierungsfertigstellungsreihenfolge festgelegt
        - Schlecht planbar
    - Top-Down (IV)
        - Integration von höchster logischer Ebene her (z.B. UI)
        - Frühe Verfügbarkeit eines Simulationsmodell, anhand dessen Defekte in der Produktdefinition identifizierbar sind
        - Späte Prüfbarkeit des Zusammenspiels mit Hardware und Basissoftware
        - Integrierte Systemkomponenten setzen Dienste niedrigerer logischer Ebene voraus (schwierig zu erstellende Testhelfer)
    - Bottom-Up (IV)
        - Integration von niedrigster logischer Ebene her
        - Testtreiber erforderlich
        - Leichteres Herstellen von Testbedingungen
        - Leichtere Interpretation von Testergebnissen
        - Späte Verfügbarkeit eines Simulationsmodells
    - Outside-in (IV)
        - Beginnt bei höchster und niedrigster logischer Ebene und integriert schrittweise in die Mitte (vermindert Nachteile von Top-Down und Bottom-Up)
    - Inside-Out (IV)
        - Beginnt bei mittlerer Ebene und integriert schrittweise nach außen (vereinigt Nachteile von Top-Down und Bottom-Up)
    - Hardest-First (IV)
        - Zuerst werden die kritischen Komponenten integriert
        - Bei jedem Integrationsschritt werden die kritischen Komponenten indirekt mitgeprüft -> beste Prüfung dieser Bereiche

Aufwandsschätzung:

- **Einflussfaktoren**
  - Quantität
    - Umfang (KLOC)
    - Komplexität (Skala 1-6)
  - Qualität
    - Nutzbarkeit
    - Integrität
    - Effizienz
    - Korrektheit
    - Zuverlässigkeit
    - Wartbarkeit
    - Testbarkeit
    - Flexibilität
  - Entwicklungsdauer
    - Mehr Mitarbeiter -> Kürzere Entwicklungsdauer -> Höherer Kommunikationsaufwand -> Geringere Produktivität
  - Kosten
  - Produktivität
    - Lernfähigkeit
    - Motivation
    - Verwendete Methoden und Werkezuge
    - Ausbildung/Vertrautheit
    - Kommunikationsaufwand
- **Basismethoden**
  - Analogiemethode
    - Methode
      - Vergleich der zu schätzenden Entwicklung mit bereits abgeschlossenen Produktentwicklungen anhand von Ähnlichkeitskriterien
    - Vorteile
      - Relativ einfache und intuitive Schätzmethode
    - Nachteile
      - Intuitive, globale Schätzung aufgrund individueller Erfahrungen, nicht übertragbar
      - Fehlende allgemeine Vorgehensweise
  - Relationsmethode
    - Methode
      - Das zu schätzende Produkt wird direkt mit ähnlichen Entwicklungen verglichen
      - Aufwandsanpassung erfolgt mit Erfahrungswerten, Faktorenlisten und Richtlinien
  - Multiplikatormethode
    - Methode
      - Das zu entwickelnde System wird in elementare Teilprodukte zerlegt, denen per Analyse vorhandener Produkte ein feststehender Aufwand zugeordnet werden kann

                - Einteilung der Teilprodukte in Kategorien mit eigenem Aufwandsmultiplikator
            - Nachteile
                - Eine umfangreiche, empirische Datensammlung und –auswertung ist benötigt
                - Permanente Überprüfung der Faktoren und Kategorienaufteilung nach technischem Fortschritt
    - Phasenaufteilung
        - Methode
            - Aus abgeschlossenen Entwicklungen wird ermittelt, wie sich der Aufwand auf die einzelnen Entwicklungsphasen verteilt wird
            - Bei neuen Entwicklungen wird eine Phase entweder vollständig abgeschlossen und analysiert oder der Aufwand im Vorfeld detailliert geschätzt. Damit wird dann auf den Gesamtaufwand geschlossen.
        - Vorteil
            - Frühzeitiger Einsatz möglich
        - Nachteil
            - Nahezu unbrauchbar durch die starke Varianz von Projekt zu Projekt
- **COCOMO II**
    - Berechnet den Aufwand in PM aus geschätzter Größe (Size), Kalibrierungskonstante (A), Kostenfaktoren (EM) und Skalierungsfaktoren (SF)

$$PM = A \cdot (Size)^{1,01+0,01 \cdot \sum_{j=1}^{5} SF_j} \cdot \prod_{i=1}^{17} EM_i$$

    - Kostenfaktoren
        - Produktfaktoren
        - Platformfaktoren
        - Personalfaktoren
        - Projektfaktoren
- **Konsens-Schätzmethoden**
    - Delphi-Methode
        - Eine Reihe von Schätzern geben anonym Schätzwerte mit Begründung ab
        - Der Moderator fasst die Ergebnisse zusammen und stellt sie vor
        - Wiederholung zur Annäherung der Werte
    - Planungspoker
        - Alle Teilnehmer wählen eine Karte mit nominalem Wert (0,1,2,8,20,… für z.B. PT) und decken sie gleichzeitig auf
        - Die höchsten und niedrigsten Schätzer geben Begründungen an
        - Wiederholung zur Annäherung der Werte
- **Verwendung**
    - Frühzeitig
        - Analogiemethode
        - Relationsmethode
        - Planungspoker
    - Mit Wissen über Einflussfaktoren
        - COCOMO II
    - Für Sammlung der Unternehmensdaten

        - Multiplikatormethode

Prozessmodelle:

- **Programmieren durch Probieren (Trial & Error)**
    - Methode
        - Vorläufiges Programm erstellen
        - Anforderung, Entwurf, Testen und Wartung danach
        - Programm anpassen/Verbessern
    - Eigenschaften
        - Schnell (?)
        - Kein „nutzloser“ Zusatzaufwand
        - Schlecht strukturiert, unsystematisch und nicht gut wartbar
        - Ungeeignet für Teamarbeit
- **Wasserfallmodell**
    - Methode
        - S.o.
    - Eigenschaften
        - Dokumentgetrieben (Fertiges Dokument nach jeder Aktivität)
        - Einfach und verständlich
        - Nicht parallelisierbar oder phasenübergreifend
        - Zwang zur Spezifikation
- **V-Modell**
    - V-Modell 97
        - Jede Aktivität hat einen eigenen Prüfungsschritt (*z.B. Grobentwurf – Systemtest, Modulimplementierung – Modultest, …)*
        - Aufteilung der Arbeiter in Submodelle (Projektmanagement, Qualitätssicherung, Konfigurationsmanagement, Systemerstellung)
    - V-Modell XT
        - Aktivitäten, Produkte und Verantwortlichkeiten werden festgelegt, jedoch ohne Reihenfolge/Phasengrenzen
        - Projektentwicklung durch „Rollen“ der Mitarbeiter, definiert mit Aufgaben, Befugnissen, Verantwortlichkeiten,… *(z.B. SW-Architekt, …)*
        - Submodelle weiter aufgeteilt in Vorgehensbausteine
    - Eigenschaften
        - Jedes Produkt durchläuft die vier Zustände „in Planung“, „in Bearbeitung“, „vorgelegt“ und „fertig gestellt“.
- **Prototypenmodell**
    - Methode
        - Erstellung eines eingeschränkt funktionsfähigen Systems (Prototyp) für anfängliche Abklärung mit / Vorführung für den Kunden
        - Danach „Wegwerfen“ des Prototypen und Entwicklung des eigentlichen Systems per anderem Modell
        - Prototypen können auch inmitten anderer Phasen verwendet werden *(z.B. Architekturprototyp in Entwurfsphase)*
- **Iteratives Modell**
    - Methode
        - Funktionalität wird Schritt für Schritt erstellt und dem Produkt „hinzugefügt“

    - Typen (Planungs-/Analysephase)
        - Evolutionär: Plane und analysiere nur den Teil, der als nächstes hinzugefügt wird
        - Inkrementell: Plane und analysiere alles und iteriere dann n-mal über Entwurfs-, Implementierungs- und Testphase
- **Synchronisiere und Stabilisiere (Microsoft-Modell)**
    - Methode
        - Organisiere alle Entwickler in kleine Teams
        - Synchronisiere und stabilisiere regelmäßig
    - Phasen
        - Planungsphase (3-12 Monate)
            - Manager identifizieren das „Wunschbild“, spezifizieren das Projekt mit den Entwicklern und stellen Zeitplan und Teamstruktur auf
        - Entwicklungsphase (3 x 2-4 Monate)
            - Zusammenarbeit von Managern (Koordination/Spezifikation), Entwicklern (Entwurf/Code) und Testern
            - 3 Teilprojekte, die in drei Meilensteinen vorgestellt und zusammengetragen werden, absteigend geordnet nach Wichtigkeit (Stabilisieren)
            - Bei jedem Meilenstein muss fehlerfreie Integration und Beseitigung aller bekannten Fehler gewährleistet sein
            - Nächtliches vollständiges Neuübersetzen aller Quellen (Synchronisieren)
        - Stabilisierungsphase (3-8 Monate)
            - Manager koordinieren Beta-Tester während Entwickler den Code stabilisieren
            - Anwendung interner und externer Tests
            - Vorbereitung der Auslieferung
    - Vorteile
        - Effektiv durch kurze Produktzyklen und kleine Teams
        - Priorisierung und Modularisierung nach Funktionen
        - Frühe Rückmeldungen
    - Nachteile
        - Nicht für alle Softwarearten geeignet
        - Mangelnder Kontakt über Teamgrenzen hinweg
- **Agile Prozesse**
    - Manifest
        - Individuen und Interaktion wichtiger als Prozesse und Werkzeuge
        - Laufende Software wichtiger als vollständige Dokumentation (Vermeiden von Dokumenten)
        - Kundenmitarbeit wichtiger als Vertragsverhandlungen (Einbeziehung des Kunden)
        - Sich auf Änderungen einstellen wichtiger als Verfolgen eines Plans (Inkrementelle Planung und Minimum an Vorausplanung, schnelle Reaktion auf Änderung)
    - Extreme Programming (XP)
        - Paarprogrammierung, Programmierrichtlinien und gemeinsamer Quelltextbesitz
            - Fahrer/Beifahrer-Prinzip (ständige Durchsicht)
            - Höhere Qualität und weniger Inspektionen

        - Nahezu doppelte Kosten -> Vorteil zweifelhaft
    - Testgetriebene Entwicklung, textuelle Beschreibung der Anwendungsfälle
        - Komponententest vor Komponentenimplementierung (inkrementeller Entwurf)
        - Verwendung von Komponenten-, Akzeptanz- und Stresstests
        - Automatisiert, häufig und wiederholbar
        - Konkretes Feedback über Erhalt der Funktionalität
        - Dienen Qualitätssicherung, Schnittstellendefinition und Modularisierung
        - Ersatz der Spezifikation/Dokumentation
    - Inkrementeller Entwurf durch Umstrukturierungen
        - Code möglichst einfach halten (nur so weit, wie Tests die Funktionalität definieren)
        - Erhöht Produktivität und Wartbarkeit
        - Möglichst wenig Kommentare, Klassen, Funktionen, …
    - Iterative Planung in kurzen Zyklen (Planungsspiel)
        - Planung von Umfang, Zeit und Kosten bis zu den nächsten Fristen (Wochenzyklus, Quartale, …)
        - Ständige Korrektur des Plans
        - Kunde trifft geschäftsrelevante, Entwicklungsteam trifft technische Entscheidungen
        - Kunde arbeitet mit -> jederzeitige Korrektur möglich
        - Auslieferungsplanung als gemeinsam erstellter Ersatz für Anforderungsanalyse (ganz am Anfang)
        - Iterationsplanung als Korrektur inmitten des Entwicklungsprozesses (für ca. 1-4 Wochen jeweils)
        - Aufgaben werden auf Karten notiert von Teammitgliedern eingeschätzt und zusammengetragen
        - Direktes Feedback vom Kunden (stände Anwesenheit/Mitarbeit in Praxis meist utopisch)
    - Kritik
        - Asymptotische Kostenkurve (laut Kent Becks) beruht auf der Erfahrung einzelner
        - Fehlende Dokumentation
        - Nicht reproduzierbarer ad-hoc Prozess
        - Kostenintensive und umstrittene Paarprogrammierung
        - Umgewöhnung/Umlernen bisheriger Mitarbeiter
        - Große Projekte kommen ohne Anforderungs-/Entwurfsphase nicht aus
        - Schlecht für große Teams geeignet
    - Eignung
        - Für kleine Teams und änderungsreiche Anforderungen
- Scrum
    - Anforderungsliste für Produktanforderungen und Liste aller Projektarbeiten, priorisiert durch Kunde
    - Unterteilung des Projekts in Abschnitte (Sprints) mit Anforderungen und Zielen und einem Review danach (Retrospektive)
    - Aufgabenliste mit Aufgabenzusammenfassung + Beschreibung aller Aufgaben für aktuelles Sprint-Ziel

- Hindernisliste mit allen Hindernissen, die vom Teamleiter (Scrum Master) mit dem Team besprochen und beseitigt werden
- Rollenteilung
  - Kunde/Auftraggeber (Priorisierung, Budget, Anforderungsanalyse)
  - Scrum Master (Organisation, Übersicht, Hindernisbeseitigung)
  - Entwicklungsteam (Aufteilung in Fachgebiete für Entwicklung)
- Tägliche Treffen mit Besprechung der geleisteten und verbleibenden Arbeit

Glossar:

SOFTWARETECHNIK: Die technologische und organisatorische Disziplin zur systematischen Entwicklung und Pflege von Softwaresystemen, die spezifizierte funktionale und nicht-funktionale Attribute erfüllen.
Antworten auf: Planung, Anforderungsermittlung, Spezifikation, Entwurf, Pflege/Restauration, Qualitätssicherung, Prozessmanagement/Organisation.

Software: Sammelbezeichnung für Programme und Daten, die für den Betrieb von Rechensystemen zur Verfügung stehen, einschl. der zugehörigen Dokumentation (Brockhaus Enzyklopädie).
Die zum Betrieb einer Datenverarbeitungsanlage erforderlichen nichtapparativen Funktionsbestandteile (Fremdwörter-Duden).

Produkt: Ein Produkt ist ein in sich abgeschlossenes, für einen Auftraggeber oder einen anonymen Markt bestimmtes Ergebnis eines erfolgreich durchgeführten Projekts oder Herstellungsprozesses.

Softwareprodukt: Produkt, das aus Software besteht.

System: Ausschnitt aus der realen und gedanklichen Welt, bestehend aus realen Gegenständen, Konzepten und deren Strukturen/Beziehungen oder einer Mischung dieser.
(z.B. Verkehrssystem, Biologische Taxonomie, Gesundheitssystem, …)

Systemkomponente: Bestandteil eines Systems, das mit anderen Komponenten in Beziehung steht und eine aufgaben-, sinn- oder zweckgebundene Einheit bildet. Besitzt Grenzen und Schnittstellen zum Rest der Welt.

Systemelement: Nicht weiter zerlegbare Systemkomponente.

Softwaresystem: System, dessen Systemkomponenten und Systemelemente aus Software bestehen.

Systemsoftware: Für eine spezielle Hardware entwickelt ermöglicht sie Betrieb und Wartung und ergänzt deren funktionale Fähigkeiten
(z.B. Betriebssysteme, Compiler, Datenbanken, …)

Anwendungssoftware: Löst durch Benutzung der Systemsoftware die Aufgaben des Anwenders mit Hilfe von Rechenanlagen

Computer-/Datenverarbeitungssystem: Anwendungssoftware + Systemsoftware + Hardware

Softwareentwicklung: Entwicklung von Software

Systementwicklung: Entwicklung eines Systems aus Hard- und Software

Funktionale Attribute: Spezifizieren die Funktionen der Software.

Nicht-funktionale Attribute: Spezifizieren, wie gut die Software ihre Funktionen erfüllt und welche inneren Qualitäten sie besitzt.
(z.B. Zuverlässigkeit, Geschwindigkeit, Benutzerfreundlichkeit/Benutzbarkeit, Sicherheit, Änderbarkeit, Dokumentationsgrad, Wartbarkeit …)

SOFTWAREFORSCHUNG: Die Bereitstellung und Bewertung von Methoden, Verfahren und Werkzeugen für die Softwaretechnik.

Methoden: Planmäßig angewandte, begründete Vorgehensweisen zur Erreichung von festgelegten Zielen. Geben an, welche Konzepte wie und wann verwendet werden und kann durch Verfahren unterstützt werden.
(z.B. Orientierung an Daten oder Funktionalität des Systems, …)

Verfahren: Ausführbare Vorschriften oder Anweisungen zum gezielten Einsatz von Methoden durch die Beschreibung konkreter Wege zum Lösung bestimmter Probleme. Enthalten formale Vorschriften und sind stark einsatzbezogen.
(*z.B. Beschreibung der Erstellung eines statischen Modells, …)*

Werkzeuge: Dienen der automatisierten Unterstützung von Methoden, Verfahren und Konzepten und erhöht durch die erzwungene Einhalten von Standards die Produktivität (Automatisierung).
(z.B. Codeüberprüfung, graphische Benutzeroberflächengestaltung, …)

SOFTWAREKONFIGURATIONSVERWALTUNG: Die Disziplin zur Verfolgung und Steuerung der Evolution von Software.

Softwarekonfiguration: Eine eindeutig benannte Menge von Software-Elementen, mit den jeweils gültigen Versionsangaben, die zu einem bestimmten Zeitpunkt im Produktlebenszyklus in ihrer Wirkungsweise und ihren Schnittstellen aufeinander abgestimmt sind und gemeinsam eine vorgesehene Aufgabe erfüllen sollen.

Software-Element: Jeder identifizierbare Bestandteil eines Produktes oder einer Produktlinie. Ist mit eindeutigem Bezeichner identifiziert, der bei Änderung des Elements selbst geändert werden muss, um eine Fehlidentifikation zu vermeiden.
(z.B. eine einzelne Datei sein, eine Konfiguration, ...)

Version: Die Ausprägung eines Software-Elements zu einem bestimmten Zeitpunkt.

Revisionen: Zeitlich nacheinander liegende Versionen.

Varianten: Alternative Versionen.
(z.B. Anpassung an andere Datenstrukturen/Algorithmen/Platformen/…)

Striktes Ein-/Ausbuchen: Bei der Versionskontrolle mit Depots darf ein Element von nur einem Benutzer ausgebucht werden. Er besitzt eine exklusive Änderungsreservierung, die erst beim Einbuchen wieder entfernt wird. Kann zu zeitlichen Verzögerungen führen, verhindert aber gleichzeitige Arbeit/Verschmelzungsaufwand.

Optimistisches Ein-/Ausbuchen: Bei der Versionskontrolle mit Depots darf ein Element von beliebig vielen Benutzern ausgebucht werden. Gleichzeitige Änderungen an gleicher Version müssen erst zusammengeführt werden, erlaubt aber die simultane Arbeit mehrerer Entwickler ohne Verzögerung.

Delta: Unterschied/Änderungs-Skript zwischen zwei Versionen eines Software-Elements, das die Überführung einer Version in die andere ermöglicht.

Vorwärts-Deltas: Bei der Versionskontrolle werden die Grundversion und die daran durchgeführten Änderungen gespeichert. Schneller Zugriff auf frühere Versionen.

Rückwärts-Deltas: Bei der Versionskontrolle werden die aktuelle Version und die Änderungen für frühere Versionen gespeichert. Schneller Zugriff auf aktuelle Versionen.

Durchführbarkeitsuntersuchung: Schriftliche Abschätzung verschiedener Gesichtspunkte, die entscheiden, ob sich das Softwareprojekt durchführen lässt.
(z.B. fachliche DF., Alternativen, personelle DF., Risiken, ökonomische DF., rechtliche DF.)

Lastenheft: Beschreibt Ziel, Einsatz und Anforderungen des zu erstellenden Systems ohne genaue Informationen über dessen Aufbau oder Implementierung

Pflichtenheft: Das Pflichtenheft definiert das zu erstellende System (oder dessen Änderungen) so voll ständig und exakt, dass Entwickler das System implementieren können, ohne nachfragen oder raten zu müssen, was zu implementieren ist. Verfeinert das Lastenheft und beschreibt nur was zu implementieren ist (nicht wie).

Modell: Abbildung der Wirklichkeit, um von Details der Realität zu abstrahieren, Schlussfolgerungen zu erleichtern, Erkenntnisse über die Vergangenheit/Gegenwart und Vorhersagen über die Zukunft zu treffen.

Menge G: Vereinigung aus allem vergangenem, gegenwärtigen und zukünftigen Substanziellem und Konzeptuellem.

Objekt: Ein für min. ein Individuum erkennbares, eindeutig von anderen Objekten unterscheidbares, lso bestimmbares Elements aus der Menge G.

Klasse: Eine (prinzipiell willkürliche) Kategorie über der Menge aller Objekte.

Exemplar: Ein konkretes Element aus einer bestimmten Klasse.

Attribut: Eine für alle Exemplare einer Klasse definierte und vorhandene Eigenschaft, die mit einem klar definierten Wert aus einer bestimmbaren, für alle gleichen Domäne hat und für jedes Exemplar unabhängig von den anderen angegeben werden kann.

Objektidentität: Die Existenz eines Objektes ist unabhängig von seinen Attributwerten. Zwei Objekte sind auch dann unterscheidbar, wenn sie die gleichen Attributwerte besitzen.

Gleichheit x-ter Stufe: x = 0: Die Objekte sind identisch, x > 0: Gleichheit 0-ter Stufe oder Gleichheit (x-1)-ter Stufe für alle Attribute.

Zustand: Solange sich ein Objekt in einem Zustand befindet, reagiert es im gleichen Kontext immer gleich auf seine Umwelt. Ändert sich der Zustand, so reagiert das Objekt in mindestens einem Kontext anders als zuvor.

Kapselungsprinzip: Der Zustand ist zwar nach außen sichtbar, er wird aber im Inneren des Objektes verwaltet (kontrollierte Änderung).

Methodensignatur: Name, Rückgabetyp und Parameterliste einer Methode.

Assoziation: Definiert Eigenschaften von n-ären Relationen zwischen (nicht zwingend diskunkten) (Mehrfach-)Mengen, die als Klassen mit Multiplizitäten angegeben werden.

Verknüpfung: Beziehung zwischen Exemplaren, die eine tatsächliche Beziehung darstellen.

Kardinalität: Anzahl der Elemente einer Menge.

Multiplizität: Ein geschlossenes Intervall der zulässigen Kardinalitäten.

Aggregation: Teil-Ganzes-Beziehung

Komposition: Strenger als Aggregation, Teile haben keine Daseinsberechtigung ohne das Ganze.

Qualifizierer: Ein(e) Attribut(kombination), die eine Partitionierung auf einer Menge der assoziierten Exemplare definiert.

Qualifizierte Assoziation: Eine Assoziation, bei der die Menge der referenzierten Objekte durch einen Qualifizierer partitioniert ist (1:n oder m:n Beziehung).

Klassenattribute/-methoden: Eigene Attribute und Methoden einer Klasse (als Objekt). Diese sind unabhängig von den Exemplaren.

VERERBUNG: Definiert Unterklassen/Spezialisierungen über die (transititve) Teilmengenbeziehung der Objektmengen, d.h. für Klassen A, B und dazugehörige Mengen O, U gilt, dass B von A erbt, falls U Teilmenge von O ist => B ist Sonderfall von A.

LISKOVSCHES SUBSTITUTIONSPRINZIP: Ein einem Programm, in dem U eine Unterklasse von K ist, kann jedes Exemplar der Klasse K durch eine Exemplar von U ersetzt werden, wobei das Programm weiterhin korrekt funktioniert.

Signaturvererbung: Eine in der Oberklasse definierte und evtl. implementierte Methode überträgt nur ihre Signatur auf die Unterklasse.

Implementierungsvererbung: Eine in der Oberklasse definierte und implementierte Methode überträgt ihre Signatur und ihre Implementierung auf die Unterklasse (d.h. keine Implementierungsvererbung ohne Signaturvererbung).

Überschreiben: Eine geerbte Methode unter Beibehaltung der Signatur wird neu implementiert.

Überladen: Eine geerbte Methode existiert parallel zu einer neuen Methode mit gleichem Namen, aber anderer Signatur.

Schnittstelle: Definition einer Menge abstrakter Methoden, die von den Klassen, die sie implementieren, angeboten werden müssen (Garantie und Verpflichtung der Implementierung). Eine Schnittstelle kann eine andere Schnittstelle erweitern (keine Vererbung).

Varianz: Modifikation der Typen der Parameter einer überschreibenden Methode.

Kovarianz: Varianz, bei der der Parametertyp in der überschreibenden Methode spezialisiert wird (parallel).
(Substitutionsprinzip => nur Kovarianz der Ausgabeparameter, stärkere Nachbedingungen)

Kontravarianz: Varianz, bei der der Parametertyp in der überschreibenden Methode verallgemeinert wird (gegenläufig).
(Substitutionsprinzip => nur Kontravarianz der Eingabeparameter, schwächere Vorbedingungen)

Invarianz: Keine Modifikation des Parametertyps in einer überschreibenden Methode.

Dynamische Polymorphie: Vielgestaltigkeit der Methoden in einem Vererbungsbaum, indem diejenige Methode mit der angegebenen Signatur aufgerufen wird, die in der Hierarchie am speziellsten ist.

Anwendungsfall: Eine typische, gewollte Interaktion eines oder mehrerer Akteure mit einem (geschäftlichen oder technischen) System (Durch Akteur initiiert und erfahrbar).

Trivialer Lebenszyklus: Liegt vor, wenn zwischen Initialisierung und Destruktion eines Objektes nur ein einziger Zustand existiert.

Starke Kohäsion: Ein über eine wohldefinierte Schnittstelle erreichtes Subsystem, das den Leser durch das Modell führt und mit einem aussagekräftigen Namen eine Menge von Klassen auf einer höheren Abstraktionsebene betrachtbar macht, welche für sich allein betrachtet und verstanden werden können (Prinzip innerhalb eines Subsystems).

Schwache Kopplung: Möglichst große Teile des Vererbungsbaumes sowie Aggregationen und Assoziationen sind gemeinsam im Subsystem vorhanden (Prinzip zwischen mehreren Subsystemen).

Softwarearchitektur: Gliederung eines Softwaresystems in Komponenten und Subsysteme, dazu Entwurf und Spezifikation dieser Bestandshierarchie.

MODUL: Ein Modul ist eine Menge von Proramm-Elementen, die nach dem Geheimnisprinzip gemeinsam entworfen und geändert werden.
(z.B. Menge von Klassen, Datenstrukturen, Prozessen, Makros, …)

GEHEIMNISPRINZIP / KAPSELUNGSPRINZIP: Jedes Modul verbirgt eine wichtige Entwurfsentscheidung hinter einer wohldefinierten Schnittstelle, die sich bei einer Änderung der Entscheidung nicht mit ändert (Black-Box-Prinzip).

BENUTZTRELATION: Programmkomponente A benutzt Programmkomponente B genau dann, wenn A für den korrekten Ablauf die Verfügbarkeit einer korrekten Implementierung von B erfordert.

BENUTZTHIERARCHIE: Eine zyklenfreie Benutztrelation (DAG).

ABSTRAKTE/VIRTUELLE MASCHINE: Eine Menge von Softwarebefehlen und –Objekten, die auf einer darunterliegenden (abstrakten oder realen) Maschine aufbauen und diese ganz oder teilweise verdecken können. Zwischen abstrakten Maschinen herrscht eine Benutztrelation.
(z.B. Java VM, Betriebssystem, Programmiersprachen, APIs, …)

Betriebssystem: Eine mächtige Systemsoftware als abstrakte Maschine, die die Hardware eines Systems verdeckt und mit Prozessverwaltung, virtuellem Speicher, Datenhaltung auf Hintergrundspeicher, Kommunikation, (graphischer Benutzeroberfläche) und Kommandosprachen einen Grund für die Anwendungssoftware bildet.

PROGRAMMFAMILIE / SOFTWARE-PRODUKTLINIE: Menge von Programmen, die erhebliche Anteile von Anforderungen, Entwurfsbestandteilen oder Softwarekomponenten gemeinsam haben (Ausnutzung von Gemeinsamkeiten und Wiederverwendung von Entwürfen und Komponenten sorgt für eine kostengünstigere und schnellere Entwicklung neuer Programme).

Allgemeines vs. Flexibles Programm: Ein Programm ist allgemein, falls es in vielen Situationen ohne Änderung benutzt werden kann (hohe Laufzeit-/Speicherkosten). Ein Programm ist flexibel, falls es leicht für viele Situationen abgeändert werden kann (hohe Entwurfskosten).

SCHICHTENARCHITEKTUR: Gliederung einer Softwarearchitektur in hierarchisch geordnete Schichten. Eine Schicht besteht dabei aus einer Menge von Software-Komponenten mit einer wohldefinierten Schnittstelle, nutzt die darunterliegenden Schichten und stellt seine Dienste darüber liegenden Schichten zur Verfügung. Zwischen den Schichten ist die Benutztrelation linear, baumartig oder ein

azyklischer Graph. Innerhalb einer Schicht (Aufteilung in Partitionen = Subsysteme) ist sie beliebig, aber meist über eine Fassade (Entwurfsmuster) gebündelt.

Transparente vs. Intransparente Schichtenarchitektur: Eine Schichtenarchitektur heißt intransparent, wenn eine Schicht nur auf Dienste der Schicht direkt unter ihr zugreifen kann. Bei eine transparenten Architektur kann eine Schicht auf alle Schichten unter ihr zugreifen.

K-Schichten-Architektur: Eine Schichtenarchitektur bestehend aus k hierarchisch geordneten Subsystemen.

K-stufige Architektur: K-Schichten-Architektur mit auf mehreren Systemen verteilten Schichten. *(z.B. kommerzielle Webservices, ...)*

ENTWURFSMUSTER: Ein Software-Entwurfsmuster beschreibt eine Familie von Lösungen für ein Software-Entwurfsproblem.

Implizite Zustandsspeicherung: Der Zustand eines Objektes kann aus den Attributwerten eines Exemplars „berechnet" werden. Die Zustandsübergangsfunktion ist implizit. Dies spart Speicherplatz, ist aber meist komplizierter und nicht immer möglich.

Explizite Zustandsspeicherung: Der Zustand eines Objektes wird in dedizierten Instanzvariablen gespeichert und kann daher eingelesen und neu gesetzt werden. Die Zustandsübergangsfunktion muss deshalb auch explizit angegeben werden. Dies spart Rechenzeit und ist offensichtlicher, ist aber meist umfangreicher.

Eingebettete explizite Zustandsspeicherung: Jede Methode „kennt" den kompletten Automaten und arbeitet kontextsensitiv (Fallunterscheidungen). Dies ist kompakter und schneller.

Ausgelagerte explizite Zustandsspeicherung: Die Zustände werden durch verschiedene Klassen dargestellt, die gleichnamige Methoden mit angepassten Implementierungen verwenden (Strategie-Muster). Dies ist meist flexibler und (bei komplexen Automaten) übersichtlicher.

MOORESCHE REGEL (aktualisiert): Verdopplung der Anzahl Prozessoren pro Chip mit jeder Chip-Generation, bei etwa gleicher Taktfrequenz.

Parallel: Gleichzeitig ablaufend, simultan (Informatik). Nebeneinander verlaufend, in gleichem Abstand (Brockhaus).

Gemeinsamer Speicher: Prozessoren haben einen Speicherbereich, den sie gemeinsam benutzen. Jeder Prozessor kann jede Speicherzelle ansprechen.

Verteilter Speicher: Jeder Prozessor hat seinen eigenen Speicher, der nur ihm zugänglich ist. Zur Kommunikation schicken sich die Prozessoren Nachrichten.

Prozess: Wird durch das Betriebssystem erzeugt und enthält Informationen über Programmressourcen und Ausführungszustand (Code-Segment, Daten-Segment, Adressraum, …). Langsamer Kontextwechsel zwischen Prozessen.

Kontrollfaden: Auszuführende Instruktionsfolge innerhalb eines Prozesses mit eigenem Befehlszeiger, Keller und Registern und Zugriff auf die gemeinsame Halde des Prozesses. Schneller Kontextwechsel zwischen Kontrollfäden.

Kritischer Abschnitt: Ein Bereich, in dem ein Zugriff auf einen gemeinsam genutzten Zustand stattfindet.

White Box Tests: Tests mit Bestimmung der Werte mit Kenntnis von Kontroll- und/oder Datenfluss.

Black Box Tests: Tests mit Bestimmung der Werte ohne Kenntnis von Kontroll- und/oder Datenfluss aus der Spezifikation heraus.

Zwischensprache: Assembler-ähnliche Sprache mit beliebigen sequenziellen Befehlen und nur bedingten oder unbedingten Sprungbefehlen zu festen Stellen der Befehlsfolge (keine Schleifen, Bedingte Blöcke, …).

Strukturerhaltende Transformation: Übertrag des Quellcodes in eine möglichst reduzierte Zwischensprache, bei der ausschließlich die Befehle, die die Ausführungsreihenfolge beeinflussen, durch Befehlsfolgen der Zwischensprache ersetzt werden, wobei die übrigen, direkt übernommenen Befehle bei gleicher Parametrisierung in derselben Ausführungsreihenfolge bleiben.

Grundblock: Maximal lange Folge fortlaufender Anweisungen der Zwischensprache, die nur am Anfang betreten und erst am Schluss verlassen wird.

Kontrollflussgraph: Darstellung eines Programms P als ein gerichteter Graph G mit den Grundblöcken als Knoten und den Sprüngen zwischen den Grundblöcken als Kanten. Jeder KFG besitzt einen Start- Stoppknoten.

Zweig: Eine Kante e im Kontrollflussgraph G.

Vollständige Pfade: Ein Pfad in einem KFG vom Start- zum Stoppknoten.

Subsumieren: Testverfahren x subsumiert Testverfahren y, wenn jede Menge von Pfaden, die x erfüllen, auch y erfüllt.

Kurzauswertung: Die Auswertung einer zusammengesetzt Bedingung wird abgebrochen, sobald das Ergebnis feststeht.
(z.B. if (i != 0 && a[i] == 1) … Falls i = 0, so wird der Rest nicht ausgewertet)

α-ω-Zyklus: Bezeichnet den kompletten Lebenszyklus eines Objektes, von der Speicherallokation (vor den Konstruktoren) bis zur Speicherfreigabe (nach den Destruktoren).

Regressionstest: Wiederholung eines bereits vollständig durchgeführten Systemtests aufgrund von Änderung/Korrektur des Systems zur Sicherstellung der weiteren Funktion (Vergleich mit vorangegangenem Test dient der Bewertung der Änderung).

Software-Sanierung: Alte Software ist der überwiegende Teil der installierten Software, woraus die Pflicht entsteht, diese zu sanieren, verstehen und in eine besser wartbare Form zu bringen, damit sie erweitert und verbessert werden kann => Restrukturiertes (erweitertes) System.

Änderungsverwaltung: Erfassung und Verwaltung eingehender Fehlermeldungen und Verbesserungsvorschlägen sowie Entscheidung über die Bearbeitung von Änderungsanträgen/Fehlermeldungen.

Änderungsverfolgung: Software zum Melden und Bearbeiten von Fehlern für mehrere Benutzer/Entwickler.
(z.B. BugZilla, JIRA, …)

LOC / KLOC: Einheit für Softwareumfang in Codezeilenanzahl
(LOC = Lines of Code, KLOC = 1000 Lines of Code)

PJ /PM /PW /PT /PS: Einheit für Softwareaufwand in Personenjahren/-monaten/-wochen/-tagen/-stunden.
(1 PJ = 9-10 PM, 1 PM = 4 PW, 1 PW = 5 PT, 1 PT = 8 PS)

Ingenieurszeit: Vereinigung aller Phasen von Definition bis Implementierung.
(Faustregel: durchschnittlich 350 getestete Quellcodezeilen pro Ingenieurmonat)