



SOFTWARE-ENGINEERING

.... MEHR ALS NUR PROGRAMMIEREN ;-)



ÜBERBLICK

- Allgemeines
- Projektmanagement & Entwicklungsmodelle
- Modularisierung & Modellierung
- Codeversionierung
- Qualitätssicherung
- Best Practices

ALLGEMEINES: HINTERGRUND & PRAXIS

WARUM?

- Komplexität von:
 - Systemen
 - Anforderungen
 - Interaktion
 - Software-Projekten
 - Digitalisierung

*"...as long as there were no machines,
programming was no problem at all;*

*when we had a few weak computers,
programming
became a mild problem,*

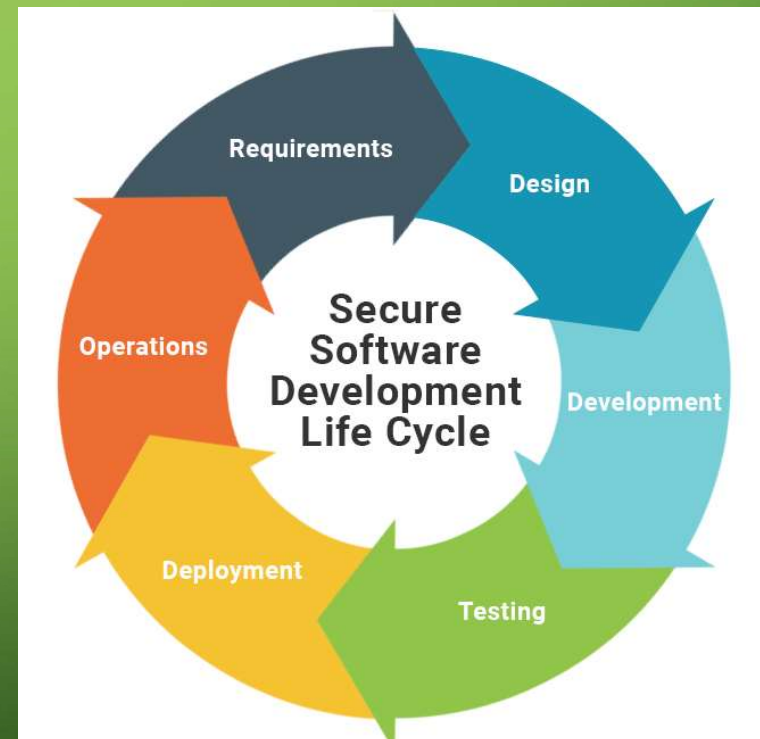
*and now we have gigantic computers,
programming has become an equally gigantic
problem."*

Edsger Dijkstra

ALLGEMEINES: HINTERGRUND & PRAXIS

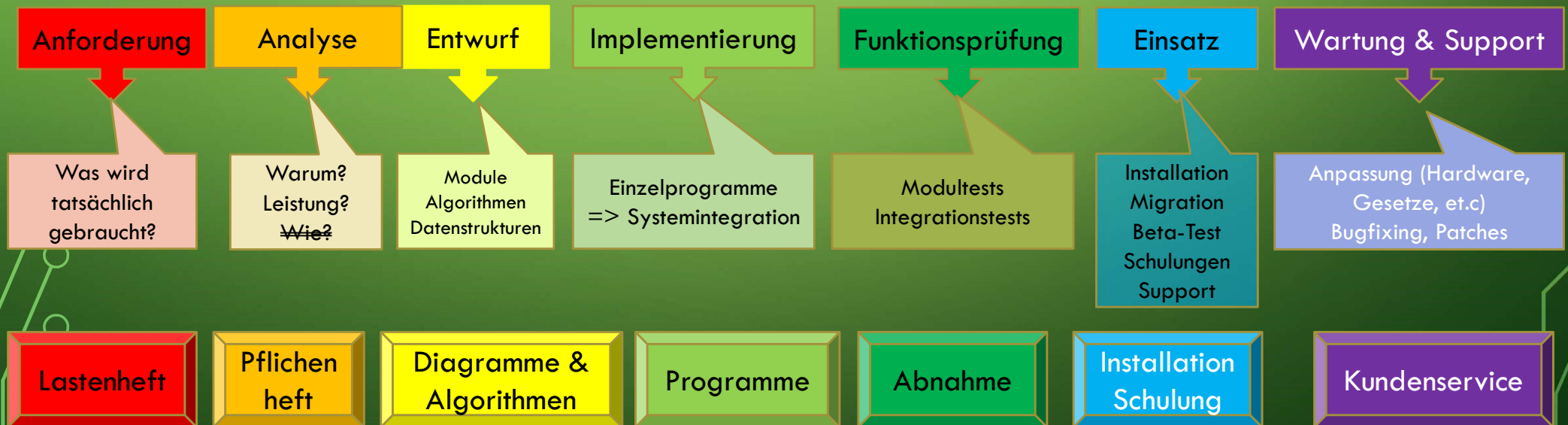
DEFINITION (IEEE)

- Prinzipien, Methoden & Werkzeuge, um Software...
 - mit einem festgelegten **Funktionsumfang**
 - in ausreichender **Qualität**
 - innerhalb eines geg. **Budget**rahmens
 - zum geplanten **Termin**
- zu erstellen.
- Zusätzliche Aufgabenbereiche:
 - Qualitätssicherung
 - Wartung
 - Nachnutzung
 - Konfigurationsmanagement
 - **Social Skills**



PROJEKTMANAGEMENT & ENTWICKLUNGSMODELLE

PROKTPHASEN ALLGEMEIN



ENTWICKLUNGSMODELLE

HINTERGRUND

- Wahl/Kombination abhängig von:
 - Dauer des Entwicklungsprozesses
 - Systemkenntnisse & Erfahrung
 - Sozialer Kontext des Entwicklungsprozesses
 - Kundenkontakt
 - Entwicklerteams

ENTWICKLUNGSMODELLE

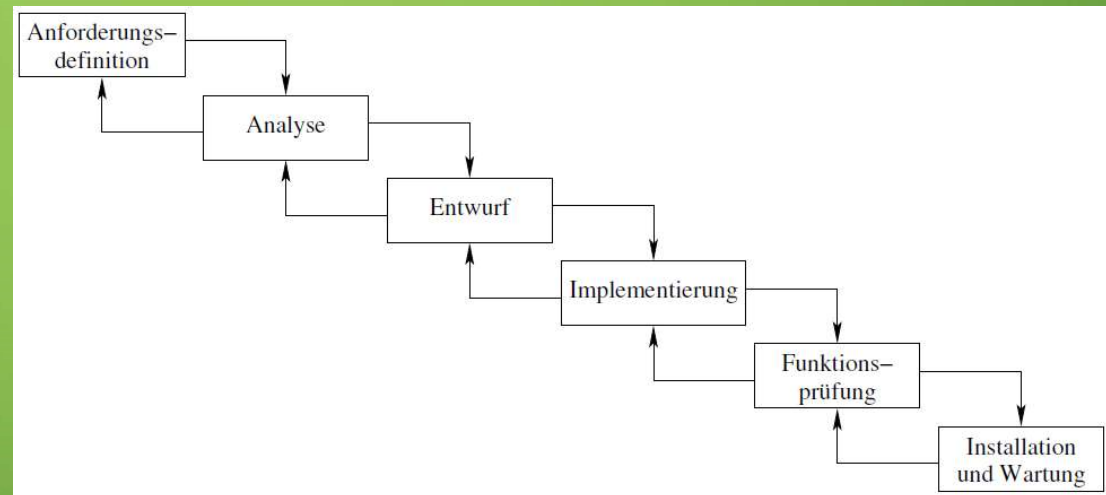
- Klassisch:
 - Wasserfall-Modell
 - V-Modell
- Agile Methoden:
 - Scrum/Kanban
 - Spiralmodell
 - Extreme Programming/Pair Programming

Planung & Budget
vs.
Anforderung & Qualität

ENTWICKLUNGSMODELLE – WASSERFALLMODELL

HINTERGRUND

- Strikte Projektphasen
 - mit Ergebnisdokumenten
 - werden sequentiell durchlaufen
- Vorteile:
 - Planungs- & Budgetsicherheit
- Nachteile:
 - Keine Rückschritt
 - ➔ nachträglich geänderte Anforderungen können nicht mehr berücksichtigt werden.



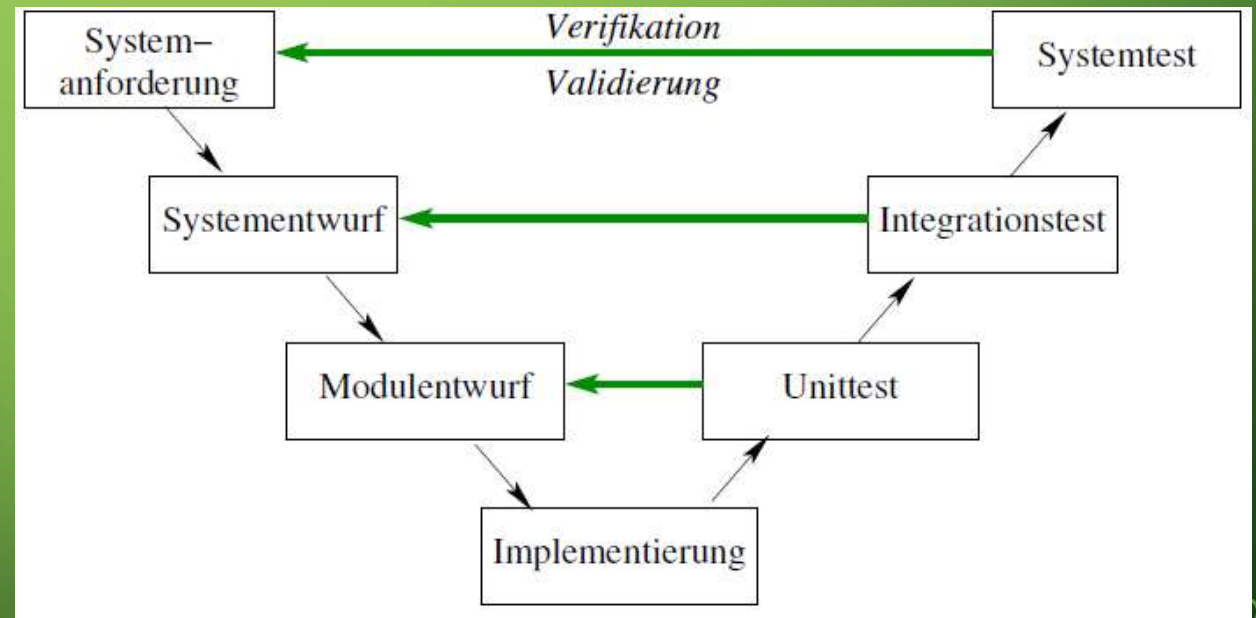
PRAKTISCHE ANWENDUNG:

- Voraussetzung:
 - Eingespielte Entwicklerteams
 - Gutes Problemverständnis
 - Erfahrung aus ähnlichen Projekten
- Modifiziertes Wasserfallmodell

ENTWICKLUNGSMODELLE – V-MODELL

HINTERGRUND

- Ähnlich Wasserfallmodell
 - Fokus Qualitätssicherung
 - ➔ Tests nach jeder Phase:
 - Verifikation
 - Validierung



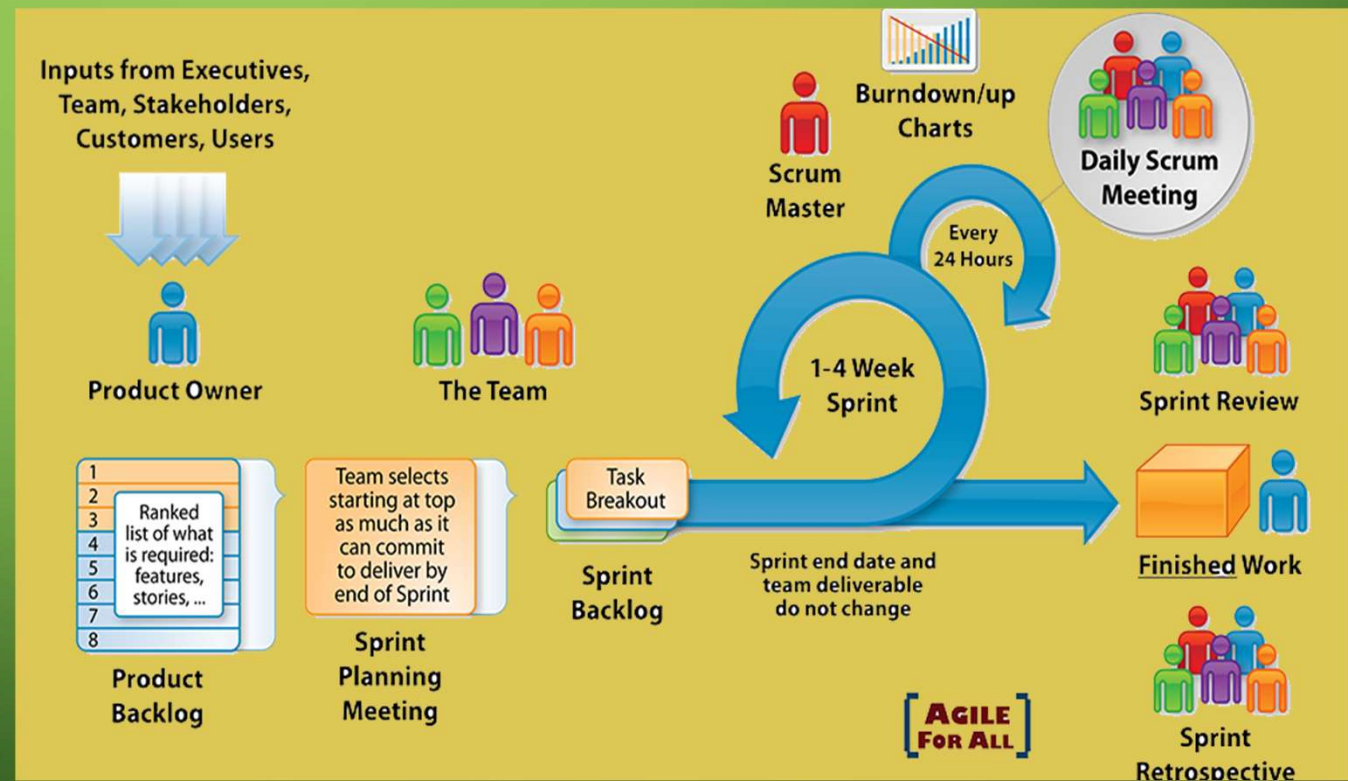
Produkt richtig
entwickelt?

richtiges Produkt
entwickelt?

ENTWICKLUNGSMODELLE – SCRUM

HINTERGRUND

- Iteratives Modell:
 - Rahmen grob vorgegeben
 - hoher Grad an Selbstorganisation
- agiles Projektmodell
 - mit fixen Time Slices
 - und fixer Rollenverteilung



ENTWICKLUNGSMODELLE – SCRUM

PROJEKTROLLEN:

- Product Owner (Kunde)

- Zielvorgabe
- Priorisierung



- Entwicklerteam:

- 5 – 10 Personen
- Aufwandsschätzung / Entwicklungsschritt
- Implementierung nächste Iteration



- Scrum Master:

- Außenstehender:
 - “Teamscoach” (Selbstorganisation)
 - Überwachung Scrumprozess
- Transparenz & Support



PROJEKTPHASEN:

- Grobplanung

- Technische Vorgaben
- Produkthanforderungen

- Gaming-Phase (Sprints)

- Selbstorganisierte Entwicklungsphasen:
 - → Software
- Vorabstimmung mit :
 - Product Owner → Sprint Planning Meeting
 - Scrum Master: → Sprint Backlog
- Daily Scrum (Standup Meeting):
- Review mit Product Owner: → Abschlussbesprechung
- Retrospective mit Scrum Master → Lessons learned

- Post-Gaming-Phase:

- Dokumentation
- Rollout
- Abnahmetests



ENTWICKLUNGSMODELLE - SCRUM VS. KANBAN

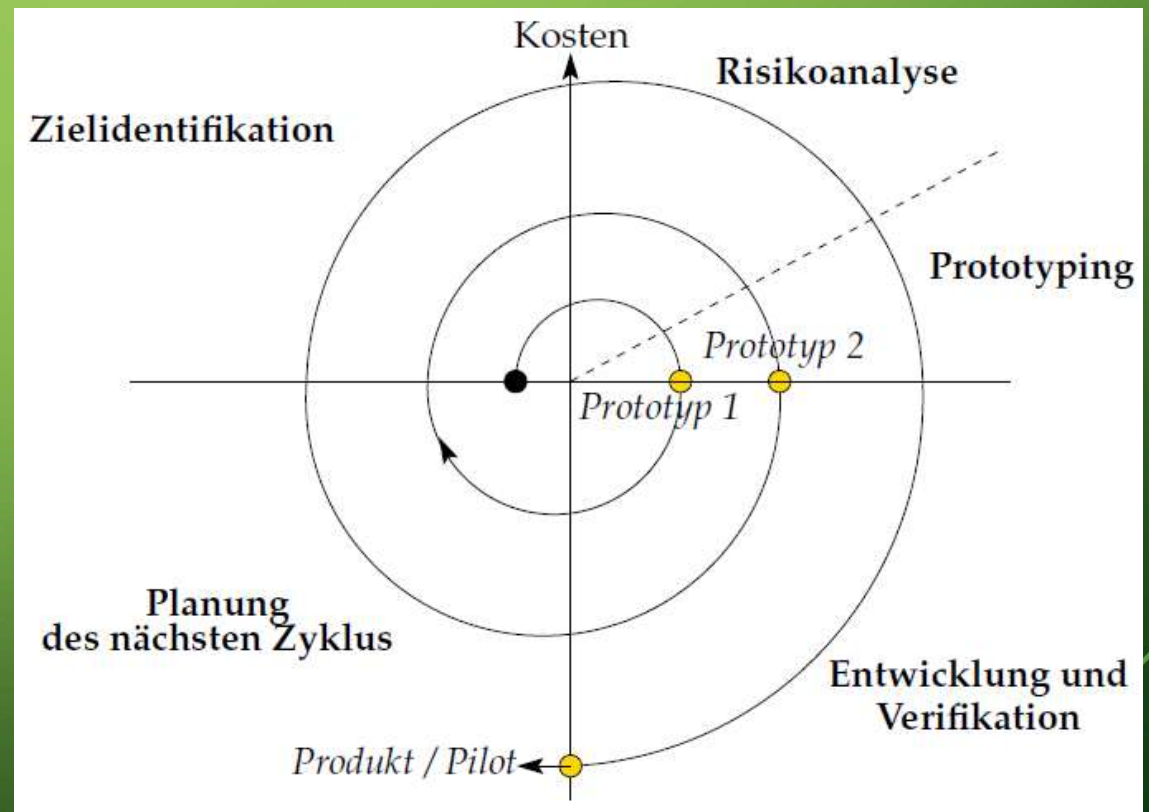
| Kanban | Scrum |
|--|--|
| Iterationen sind optional. Es kann unterschiedliche Takte für Planung, Releases und Prozessverbesserung geben. | Iterationen mit gleichen Längen sind vorgeschrieben. |
| Commitments sind optional. | Das Team commitet sich auf ein Sprint Ziel. |
| Die Durchlaufzeit (<i>Cycle Time</i>) wird als Basis-Metrik für Planung und Prozessverbesserung verwendet. | Die Team-Geschwindigkeit (<i>Velocity</i>) ist die Basis-Metrik für Planung und Prozessverbesserung. |
| Cross-funktionale Teams sind optional. Experten-Teams sind erlaubt. | Cross-funktionale Teams sind vorgeschrieben. |
| Keine Vorschrift bezüglich der Größe von Anforderungen. | Anforderungen müssen so aufgeteilt werden, dass sie sich innerhalb einer Iteration erledigen lassen. |
| WiP wird direkt limitiert. | WiP wird indirekt limitiert (durch die Menge an Anforderungen, die in einen Sprint passt). |
| Schätzungen sind optional. | Schätzungen sind vorgeschrieben. |
| Neue Anforderungen können zu jedem Zeitpunkt an das Team gegeben werden, falls Kapazitäten frei sind. | Während eines laufenden Sprints können keine neuen Anforderungen an das Team gegeben werden. |
| Gibt keine Rollen vor. | Schreibt drei Rollen vor (<i>Product Owner</i> , <i>Scrum Master</i> , Team). |
| Ein Kanban-Board kann von mehreren Teams und/oder Einzelpersonen geteilt werden. | Ein Sprint Backlog gehört einem einzelnen Team, das Produkt Backlog kann zu mehreren Teams gehören. |
| Ein Kanban-Board wird kontinuierlich weitergepflegt. | Das Sprint-Backlog wird nach jedem Sprint gelöscht und neu aufgesetzt. Das Produkt-Backlog wird kontinuierlich weitergepflegt. |
| Priorisierung ist optional. | Schreibt vor, dass alle Einträge im <i>Produkt Backlog</i> priorisiert sein müssen. |

Quelle: [https://de.wikipedia.org/wiki/Kanban_\(Softwareentwicklung\)#Scrum](https://de.wikipedia.org/wiki/Kanban_(Softwareentwicklung)#Scrum)

ENTWICKLUNGSMODELLE – SPIRALMODELL

HINTERGRUND

- Prozessorientiertes Modell
 - offenes Ende möglich
 - z.B. Betriebssysteme, Browser, etc.
 - Keine Trennung Entwicklung-Wartung
- Risikogetriebenes Modell
 - ➔ Fokus auf Risikominimierung
- Phasenmodell:
 - Phasenzyklen mehrfach durchlaufen
 - Zieldefinition aus Basis der Ergebnisse des vorherigen Zyklus.
 - pro Phase alle Schritte durchlaufen



ENTWICKLUNGSMODELLE – SPIRALMODELL

PHASENSCHRITTE

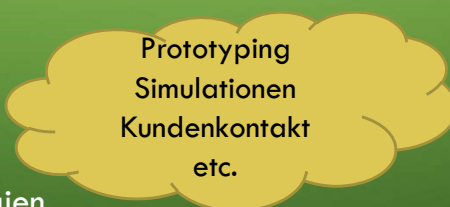
Weniger Planungssicherheit – komplexeres Projektmanagement

- (1): Zielidentifikation:

- Ziele des nächsten Zyklus
- Alternative Entwürfe
- Randbedingungen (Kosten, Zeit, etc.)

- (2) Risikoanalyse:

- Evaluierung der Entwürfe gem.
 - Randbedingungen
 - Zielen
- Entwicklung von:
 - Kosteneffizienten Strategien
 - zur Risikoüberwindung



Prototyping
Simulationen
Kundenkontakt
etc.

- (3): Entwicklung & Verifikation:

- Softwareentwurf
- Implementierung
- Tests

- (4) Planung & Evaluierung:

- Review aktueller Zyklus
- Planung nächster Zyklus:
 - Ressourcen-abhängig
 - ev. Aufteilung zur parallelen Entwicklung
- Commitment nächster Zyklus

ENTWICKLUNGSMODELLE – XP

HINTERGRUND

- Prozessorientiertes Modell

- offenes Ende möglich
- Kleinere Teams (max. 15)
- Fokus: Einfachheit
- **Sozialkompetenz!!!**

- Lauffähiger Stand jederzeit:

- ev. nicht alle Features verfügbar
- laufende Systemintegration (mind. 1 /Tag)
- laufend (automatisierte) Tests



"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß [CC BY-NC-ND](#)

- Kundenintegration:

- Kundenvertreter vor Ort
- Regelmäßige Releases:
 - mind. alle 3 Monate

- Entwicklungsprozess:

- Kurze (Spiralmodell)-Zyklen
 - ca. 3 Wochen
- Pair Programming
- Basis: User Stories
- Testfall-Spezifikation vor Entwicklung

MODELLIERUNG

HINTERGRUND

- Abbildung/Planung:

- Anforderung:**

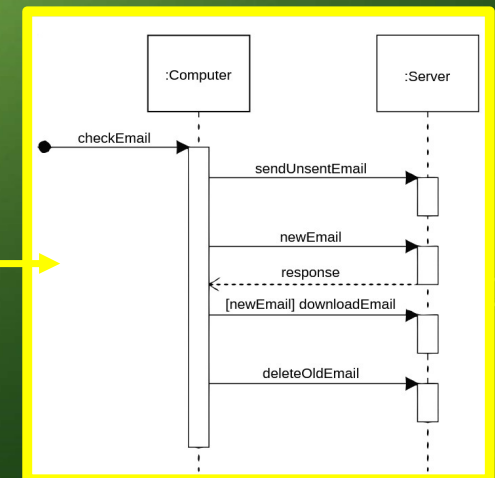
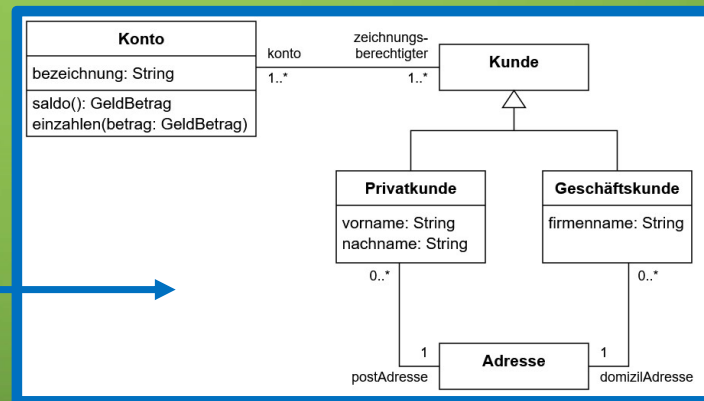
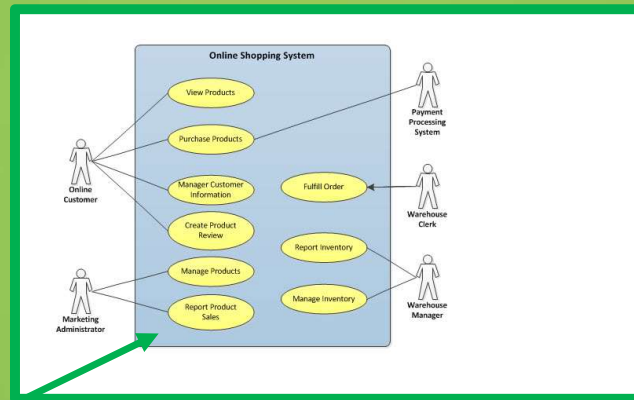
- Wer macht was womit?
 - z. B. Use Case Diagramm

- Struktur**

- Wie hängen die einzelnen Komponenten/Module zusammen?
 - z.B. UML-Diagramm

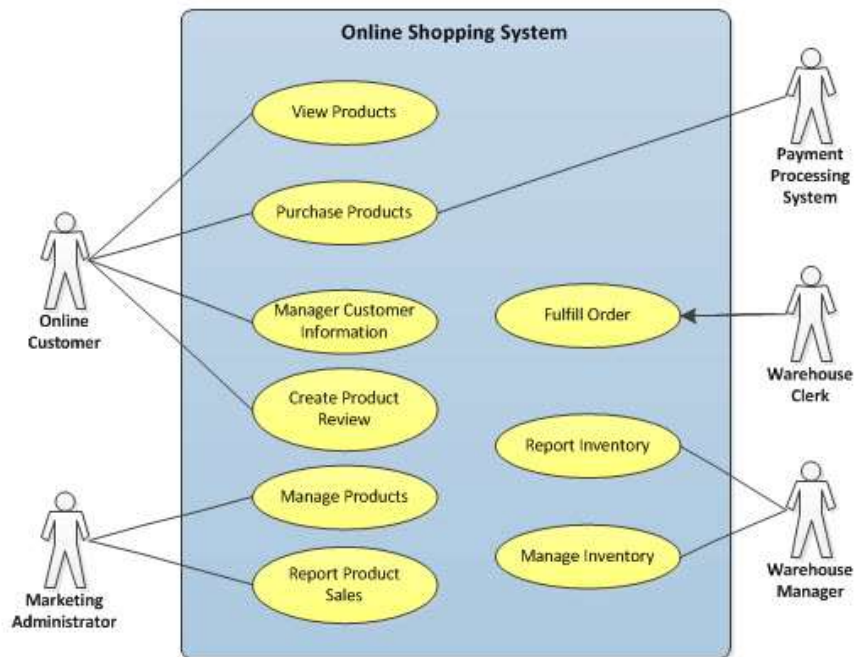
- Ablauf:**

- Welche möglichen Ausführungspfade gibt es?
 - Wie bedingen sie sich?
 - Sequenzdiagramm



MODELLIERUNG

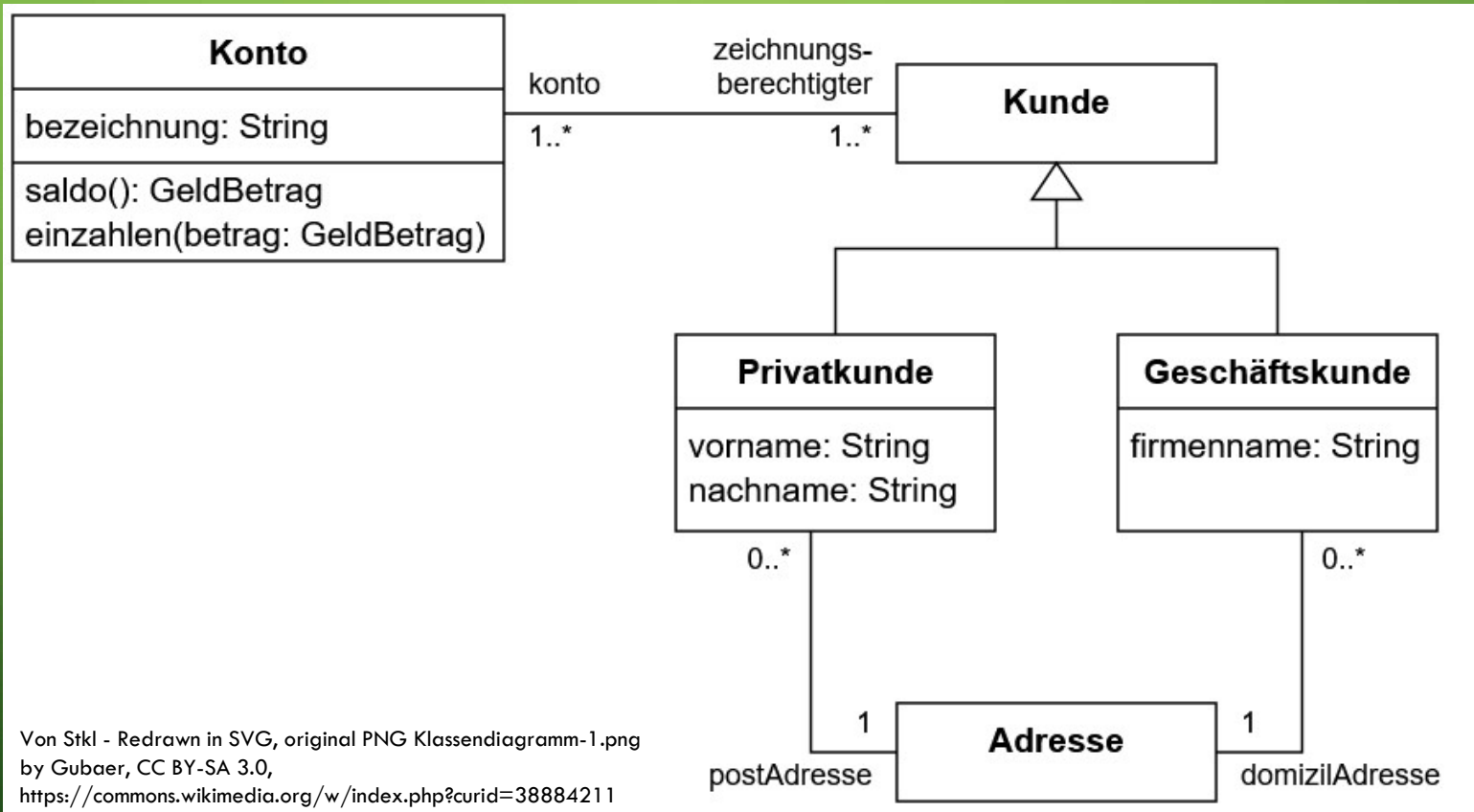
USE CASE DIAGRAMM



Tlockman, CC BY-SA 3.0 <<https://creativecommons.org/licenses/by-sa/3.0/>>, via Wikimedia Commons

MODELLIERUNG

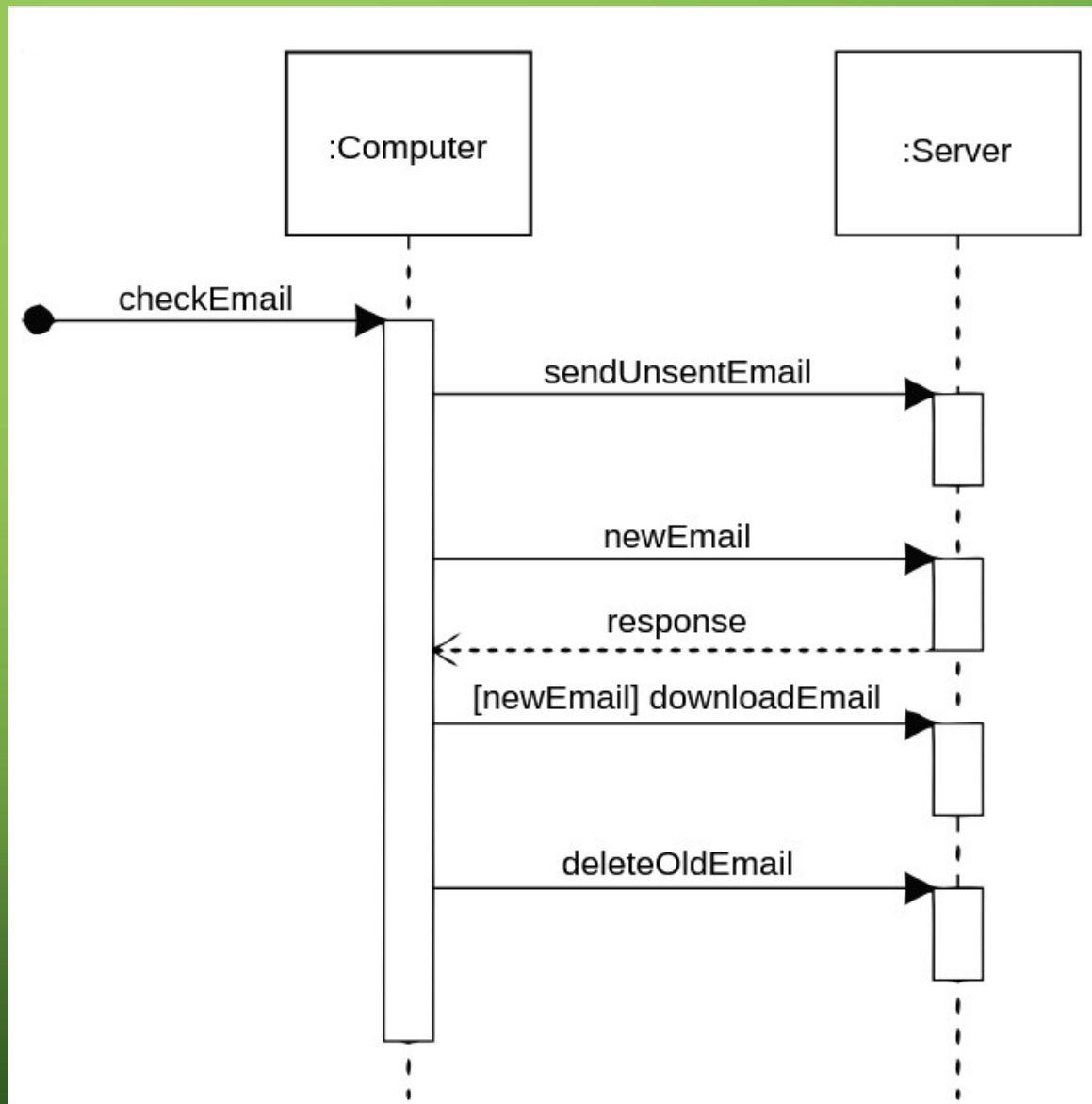
UML-DIAGRAMM



Von Stkl - Redrawn in SVG, original PNG Klassendiagramm-1.png
by Gubaer, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=38884211>

MODELLIERUNG

SEQUENZ-DIAGRAMM

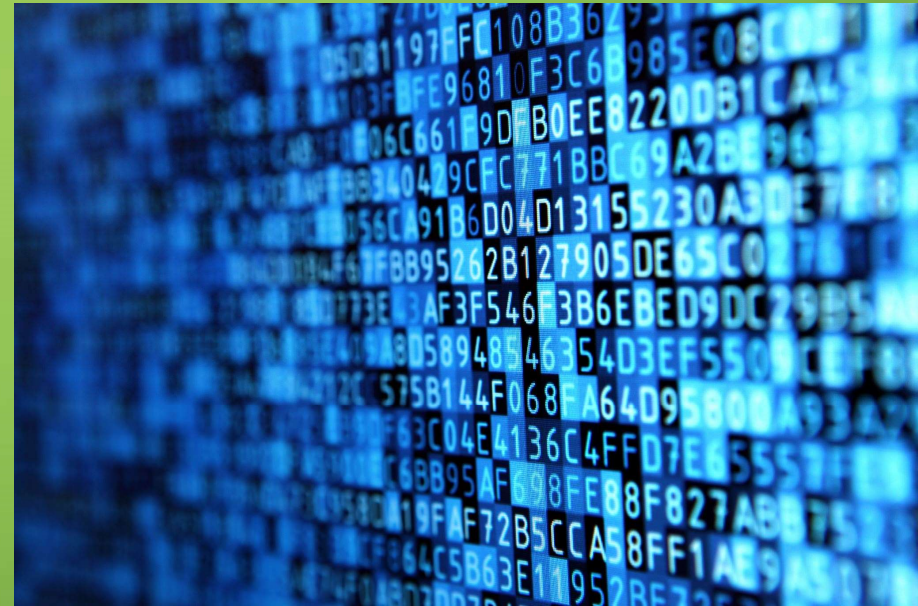


By Coupling_loss_graph.svg - File:CheckEmail.png, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=20544977>

QUALITÄTSSICHERUNG

HINTERGRUND

- Was?
 - Usability:
 - Bedienbarkeit
 - Gebrauchstauglichkeit
 - Test:
 - Ausführung um Fehler zu finden.
 - Fehler = Abweichung vom spezifizierten Verhalten
 - → keine Spezifikation, kein Fehler
 - Review:
 - z. B. Code-Review



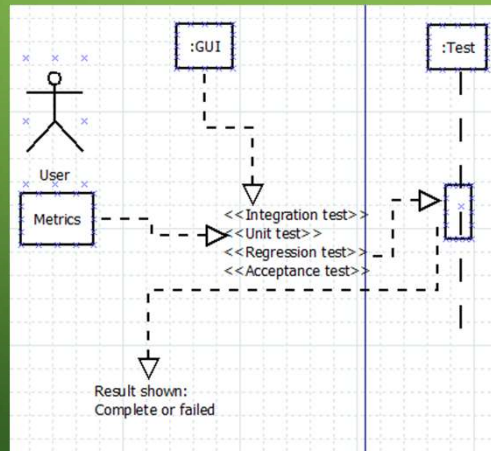
- Abnahmentest:
 - zwischen Auftraggeber und –nehmer
 - um zu verifizieren, dass die Software, das Vereinbarte leistet.

QUALITÄTSSICHERUNG

TESTS:

- Basis: Testfälle

- Welche Ereignisse werden
- bei einem bestimmten Eingabe
- produziert?



- Verfahren

- Ablaufbezogene Tests:
 - Code Coverage
- Datenbezogene Tests
 - große Mengen (zufälliger) Daten
 - => Stresstest
- Funktionale Tests:
 - Test der Funktionalität
 - lt. Spezifikation
 - ➔ UNIT-Tests
- Regressionstests:
 - Testdatenbank
 - vor jedem Release/Fix durchgespielt
 - ➔ **Zeitintensiv!!!**

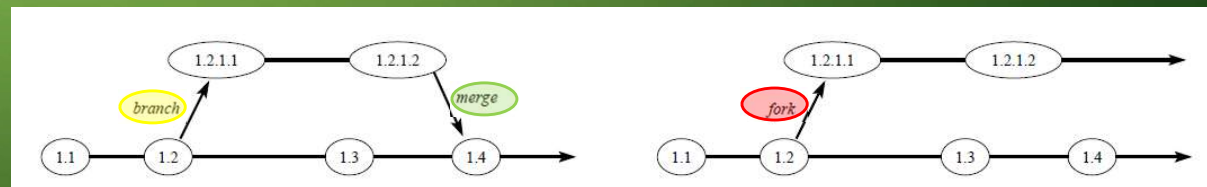
VERSIONIERUNG

HINTERGRUND

- **Grundgedanken:**
 - Teamwork am selben Code
 - Code-Historie
 - Rücksetzen auf (lauffähige) Version
- **Grund-Elemente:**
 - **Branching:**
 - Kopie des aktuellen Standes zur individuellen Weiterentwicklung (z.B. Feature, Bugfix, etc.)
 - **Merging:**
 - Zusammenführen mit eines Branches mit dem Hauptzweig des Projektes (HEAD, Trunk)
 - **Fork:**
 - Kopie des aktuellen Standes zur Weiterentwicklung ohne geplanten Merge → eigenes (Unterprojekt)

CODE-VERSIONIERUNG

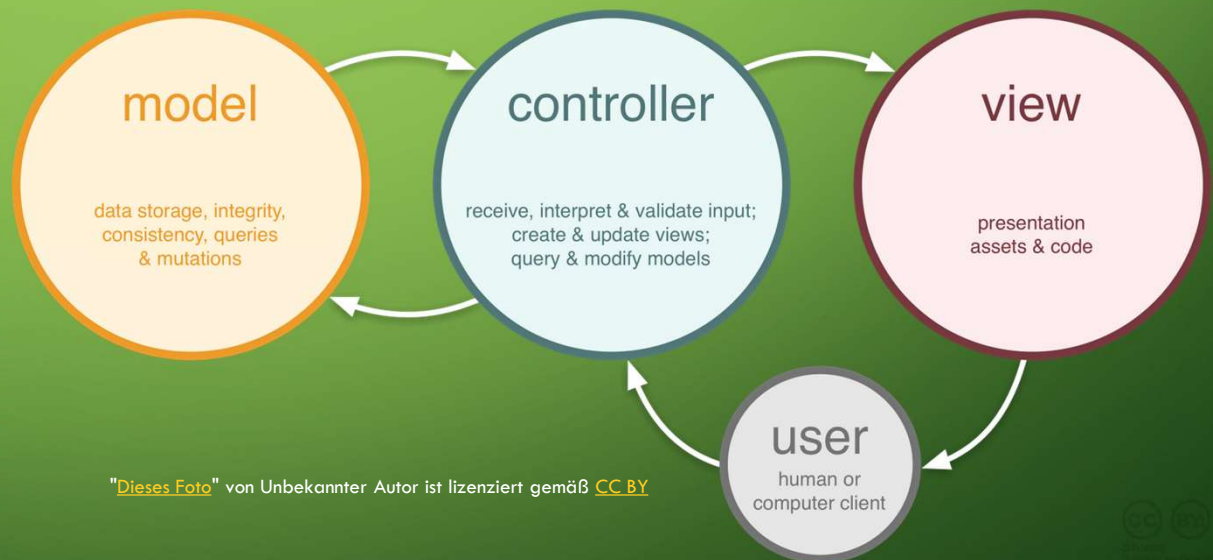
- **Modelle:**
 - Client-Server-Architektur
 - Host-/Mainframe-System
- **Tools:**
 - Git/Github
 - CVS
 - Subversion



BEST PRACTICES

GUI-PROGRAMMIERUNG

- Oberflächen-Design:
 - Usability
 - Framework
 - Optik
- Standardmodell MVC:
 - Model: Daten
 - View: Anzeige
 - Controller: Interaktion



BEST PRACTICES

UNIX SYSTEMPROGRAMMIERUNG

- Grundannahme:
 - Der Benutzer kann mit seinem Gerät umgehen
 - Anstatt ihn daran zu hindern, etwas Falsches zu tun...
 - soll er den vollen Leistungsumfang der Anwendung ausschöpfen können
 - **K**ee**p** **I**t **S**imple **S**tupid (KISS):



"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß [CC BY-NC](#)

Leitsätze:

- "Klein ist schön."
- "Jedes Programm soll genau EINE Sache GUT machen."
- "Erstelle so schnell wie möglich einen Prototyp."
- "Portabilität geht vor Effizienz."
- "Speichere numerische Daten in ASCII-Dateien."
- "Nutze die Hebelwirkung von Software zu Deinem Vorteil."
- "Vermeide Oberflächen, die den Benutzer gefangen halten."

BEST PRACTICES

CODING CONVENTIONS

- Richtlinien zur Source-Code-Erstellung:
 - Nomenklatur:
 - Variablen, Funktionen
 - Packages, Module, Header etc.
 - Formatierung:
 - Klammern
 - Einrückungen
 - etc.
 - Design Patterns (Gamma)

**Firmen- bzw. Projektabhängig
→ COMMITMENT**



"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß [CC BY](#)

FINANZIELLE ASPEKTE

- Lizenzen:
 - eigenes Lizenmodell
 - Lizenzgebühren für verwendete Komponenten
- Abrechnung:
 - Vorab-Planung inkl. Kostenabschätzung
 - laufende Zeitaufzeichnung
 - angemessene Stundensätze
 - korrekte Rechnungsstellung

QUELLEN:

- Software Engineering - *Prof. Dr. Harmut Fritzsche, Sächs. Verwaltungs- & Wirtschaftsakademie*
- Software Engineering — *Andreas de Vries, Fachhochschule Süd-Westfalen.*
- Skriptum softwaretechnik – *Herbert Klaeren, Universität Tübingen.*