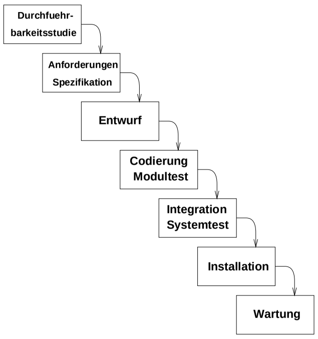
3 Software-Lebenszyklus

- früher: Code-and-Fix-Zyklus; geeignet für Ein-Personen-Projekte und kleinere Aufgaben
  1. Code-Erstellung,
  2. Verbesserung (Fehlerbeseitigung, Erweiterung des Funktionsumfanges, Effizienz, ...)
  3. gehe wieder zu 1.
- wenn Problem klar spezifiziert und eine Person die Implementierung bewältigen kann, Vorgehen angemessen
- jedoch: Wartbarkeit und Zuverlässigkeit nimmt hier kontinuierlich ab; Schwierigkeiten wenn Programmierer kündigt; heutige Projekte umfassen mehrere Personenjahre; heutige Anwender verlangen bessere Qualität; wenn Entwickler und Anwender nicht mehr identisch, oft Meinungsverschiedenheiten über den erwarteten bzw. realisierten Funktionsumfang
- Software-Krise führte zu strukturierten (Software-Entwicklungs-)Prozeßmodellen

  Vorteile: Anleitung für Entwickler, welche Aktivitäten als nächstes anstehen; Entwicklungsprozeß handhabbar durch Projektplanung, Kostenkontrolle, Abnahme von Zwischen- und Endprodukten

3.1 Wasserfallmodell



Entwicklungsprozeß in Phasen zerlegt; Phase muß abgeschlossen sein, bevor die nächste beginnt; Phase produziert Produkt (Dokument oder Programm); Unterphasen

3.1.1 Durchführbarkeitsstudie

- soll Kosten und Ertrag der geplanten Entwicklung abschätzen; dazu grobe Analyse des Problems mit Lösungsvorschlägen, Alternativlösungen; für jeden Lösungsvorschlag benötigte Resourcen, Entwicklungszeit und Kosten schätzen; Ergebnisse in einem Dokument festhalten; dieses Dokument häufig Grundlage eines Angebotes an einen potentiellen Kunden
- Problem: oft hoher Zeitdruck und begrenzte Resourcen; ungenaue Schätzungen; in bestimmten Fällen kann Durchführbarkeitsstudie auch fehlen

3.1.2 Anforderungsanalyse und Spezifikation

- exakt festgelegen, was die Software leisten soll, aber nicht, wie diese Leistungsmerkmale erreicht werden; alle relevanten Funktions- und Qualitätsmerkmale spezifizieren; diese Angaben im Pflichtenheft dokumentieren; Pflichtenheft Bestandteil des Vertrages mit dem Kundem
- Anforderungsdefinition muß verständlich, präzise, vollständig und konsistent sein

  sehr geeignet: formale Spezifikationen
- Problem: Kunde versteht sie nicht; deshalb häufig Verwendung von semiformalen Methoden, z.B. Entity-Relationship Diagramme, Datenflußdiagramme oder natürliche Sprache
- häufig vorläufige Form der Benutzeranleitung bereits Bestandteil des Pflichtenheftes

3.1.3 Entwurf

- Festlegung der Systemarchitektur: welche Module gibt es, welche Beziehungen bestehen zwischen ihnen
- häufiges Vorgehen: Top-Down Entwurf; Zerlegung des Systems in Komponenten, Zerlegung der Komponenten in Unterkomponenten usw.
- oft Verwendung spezieller Architektur- oder Systembeschreibungssprachen sinnvoll
- Systemarchitektur in Entwurfsbeschreibung festhalten

3.1.4 Codierung und Modultest

- eigentliche Implementierungs- und Testphase; früher die einzige Phase
- oft Verwendung von Programmierrichtlinien, z.B. Programmlayout, Namenskonventionen, Kommentierkonventionen; u.U. fragwürdig, denn veraltete Konventionen können Qualität reduzieren und Kreativität einschränken
- Beispiel: in einer Abteilung einer Automobilfirma Variablennamen zentral vergeben und durchnumeriert nach dem Schema: für Modul x darf man die Variablen VW1344 bis VW1576 verwenden (Beispiel real-existierend)

  Beispiel: in der Eisenbahnleittechnik-Abteilung eines Elektrokonzerns sind rekursive Prozeduren verboten, da sie zu unsicher sind; ferner muß für jede Schleife eine maximale Zahl von Durchläufen angegeben und durch einen Zähler zur Laufzeit überprüft werden (Beispiel real-existierend)
- Ergebnis der Phase: getesteter Programmtext der einzelnen Module

3.1.5 Integration und Systemtest

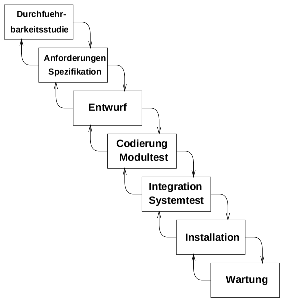
- Module zu einem Programm zusammenbinden; Zusammenspiel der einzelnen Komponenten testen; kann mit der vorangegangenen Phase verschmolzen sein; schließlich gesamtes System testen; zunächst nur innerhalb der Entwicklungsorganisation (Alpha-Test); Ergebnis der Phase: laufendes System

3.1.6 Installation und Wartung

- häufig zwei Phasen: (A) nur Auslieferung an vertrauenswürdige, ausgewählte Kunden; Ziel: Rückmeldungen, Erfahrungen, Fehlermeldungen der Benutzer (Beta-Test); (B) Auslieferung an alle Kunden
- Wartungskosten machen 60% der gesamten Softwarekosten aus; davon wiederum
  20% Fehlerbeseitigung, 20% Adaption (z.B. Anpassung an neues Betriebssystem), 50% Perfektion (z.B. Umstellung von alphanumerischer auf graphische Schnittstelle)
- andere Quelle (Swanson 1980), Studie an 400 Softwareprojekten:
  42% Änderungen der Anforderungsdefinition, 17% Änderung der Dateiformate, 12% Notfalldebugging, 9% normales Debugging, 6% Änderungen der Hardware, 5% Verbesserungen der Dokumentation, 4% Verbesserungen der Efizienz
- Wartung umso teurer, je mehr die frühen Phasen davon betroffen; am teuersten nachträgliche Änderungen der Anforderungen

3.2 Wasserfallmodell mit Rückkopplung

häufig strenge Trennung der Phasen (Idealfall) nicht erreichbar; deshalb: Möglichkeit vorsehen, in frühere Phasen zurückzukehren etwa wenn Fehler oder Inkonsistenzen entdeckt wurden



3.3 Evolutionäres Modell

- Probleme mit dem Wasserfallmodell: am Projektanfang sind nur ungenaue Einschätzungen möglich; Pflichtenheft, egal wie sorgfältig erstellt, kann nie Umgang mit dem fertigen System ersetzen; dies für viele Kunden ein Problem; es gibt Kunden, mit denen man keine präzisen Pflichtenhefte erstellen kann, weil sie nicht wissen was sie wollen; die endgültigen Anforderungen werden erst nach einer gewissen Experimentierphase deutlich
- Antizipation des Wandels von Anforderungen überhaupt nicht unterstützt; stattdessen werden die Anforderungen frühzeitig eingefroren; am Ende jeder Phase ein Dokument abliefern; dies kann zu einer Papierflut und ausufernder Bürokratie führen, ohne daß Qualität profitiert
- evolutionäres Vorgehen: Produkt als Sequenz von Approximationen realisiert; Approximation entsteht durch Änderungen bzw. Erweiterungen aus der vorangegangenen Version; einige oder sogar alle Phasen des Lebenszyklus können evolutionär realisiert werden
- Prototyping: Software-Prototypen sind Approximationen an das endgültige System mit reduziertem Funktionsumfang, reduzierter Benutzerschnittstelle bzw. eingeschränkter Performance

Spezialfälle des Prototyping

- Rapid Prototyping: Verwendung von Generatoren und ausführbaren Spezifikationssprachen

  Vorteil: sehr schnelle Realisierung, frühzeitige Validierung der funktionalen Spezifikation

  Nachteil: u.U. ineffizient, schlechte Benutzerschnittstelle

- horizontaler Prototyp: nur eine Systemschicht, z.B. hohle Benutzerschnittstelle ohne echte Funktionalität

  Vorteil: frühzeitiges Einbinden der zukünftigen Benutzer; Validierung der Spezifikation

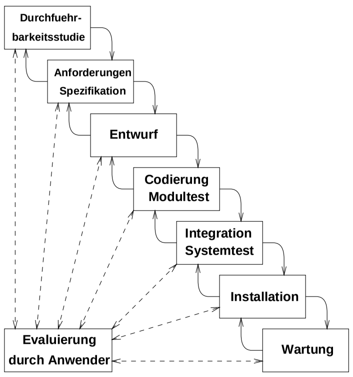
  Nachteil: nur für Demos benutzbar

- vertikaler Prototyp: abgemagerter Funktionsumfang durch alle Ebenen implementiert

  Vorteil: frühzeitige Validierung technischer Systemeigenschaften; frühzeitige Auslieferung eines Kernsystems

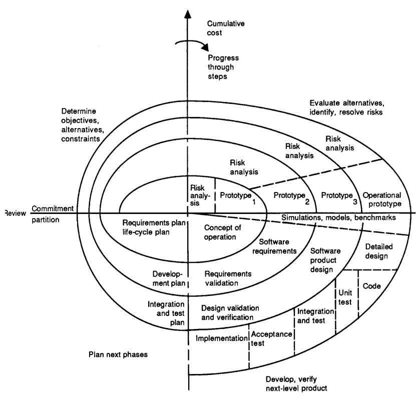
  Nachteil: teuer

allgemeines evolutionäres Modell



3.4 Spiralmodell

- Metamodell; umfaßt evolutionäre Aspekte und Risikobewertung; jeder Umlauf der Spirale entspricht dem nächsten Prototyp; Winkel $\approx$ Zeit, Radius $\approx$ Kosten
- jeder Umlauf zerfällt in vier Hauptphasen (Quadranten): Anforderungsdefinition; Bewertung von Alternativen und Risiken; Entwurf, Implementierung und Test; Auswertung und Planung des nächsten Umlaufs



3.5 Transformationsmodell

- Programm (halb-)automatisch aus formaler Spezifikation erzeugt; Anwendung von korrektheitserhaltenden Transformationen; Maschinenunterstützung
- Beispiel: Ausgangspunkt nicht-deterministische funktionale Spezifikation; zusätzliche Entwurfsentscheidungen beseitigen Nichtdeterminismus; Transformation in tail-rekursive Form; Transformation in prozedurales Programm
- Vorteil: Korrektheit garantiert

  Nachteil: anspruchsvoll, nur für mittel-große Beispiele bisher angewendet