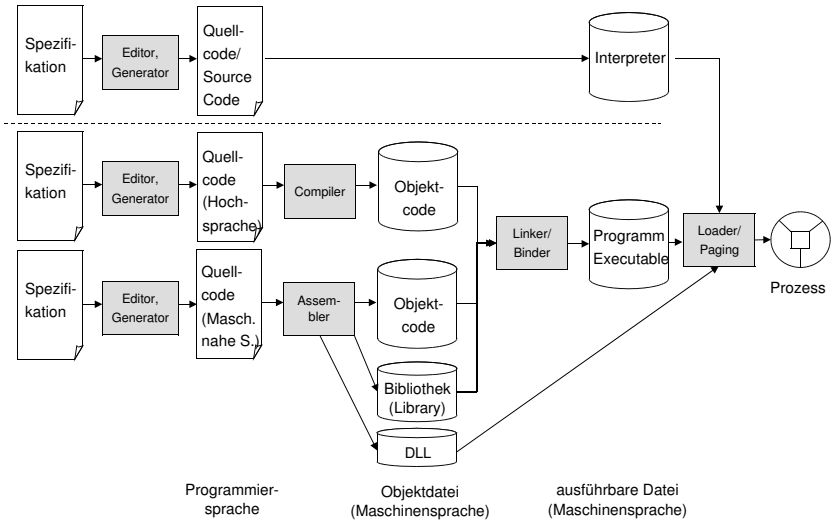
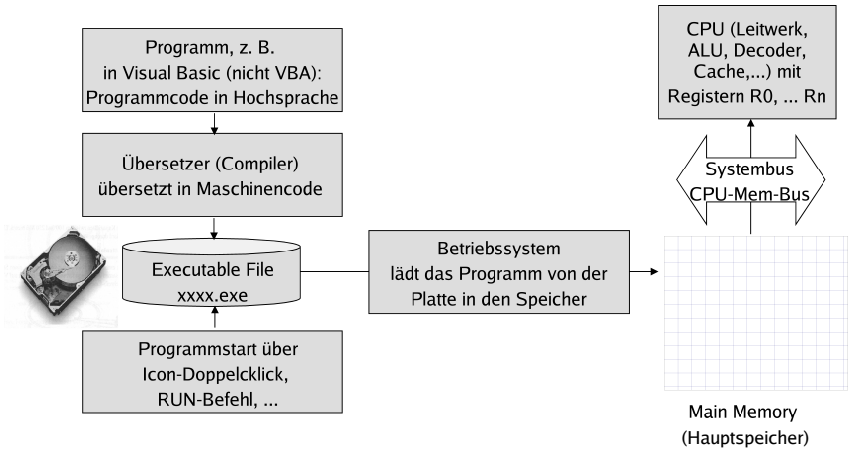
6. Software-Entwicklung & Projekt-Management

Die Folien zu Kapitel 6 habe ich größtenteils von den Kollegen Teich, Havel, Schönecker und Lehner übernommen.

Schritte der Programmentwicklung



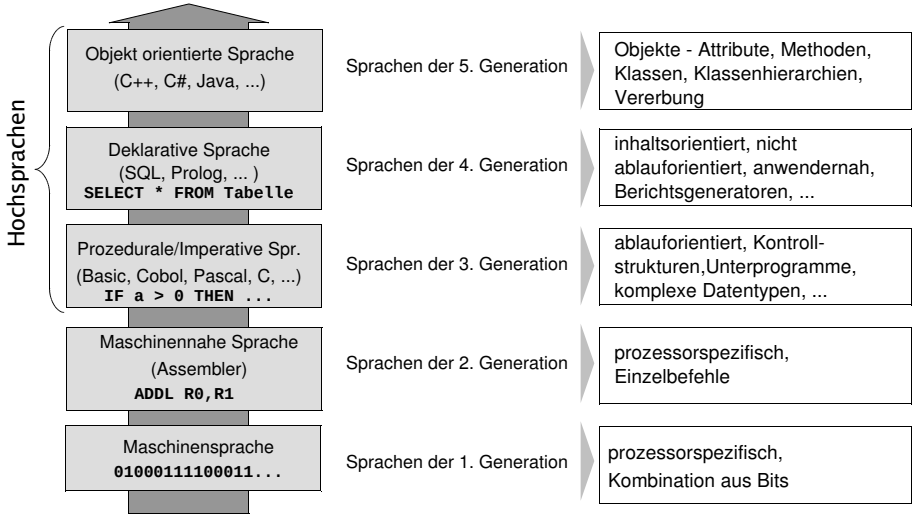
Vom Programm zur Ausführung



Compiler, Interpreter & Co.

- Höhere Programmiersprachen benötigen einen Übersetzer:
  - **Interpreter:** liest jede Anweisung eines Programms einzeln und führt diese sofort aus („interpretiert“ den Code)
  - **Compiler:** übersetzt ein komplettes Programm in eine Datei mit Maschinensprache
  - **Assembler:** übersetzt Assembler-Code in Maschinensprache
- **Linker / Binder**: Zusammensetzen von übersetzten Programmteilen
- **Debugger**
  - Unterstützung bei der Fehlerentdeckung und -beseitigung
  - verfolgt Schritt für Schritt die Ausführung des zu testenden Programms

Generationen von Programmiersprachen



Compiler vs. Interpreter (1/2)

- Kompiliertes Programm liegt in Maschinensprache vor – das, was der Computer sofort und ohne Hilfe versteht: sehr schnelle Ausführung
- Kompiliertes Programm verbraucht nur sehr wenig Speicher (auf Platte und auch im RAM)
- Compiler kann Code optimieren
  - für die jeweilige „Plattform“ (CPU)
  - allgemein – z. B. sinnlose Schleifen oder Berechnungen mit Konstanten erkennen
- Bei jeder Programmänderung ist eine neue vollständige Übersetzung nötig

Compiler vs. Interpreter (2/2)

- Interpreter ist flexibler als Compiler; z. B.
  - keine Festlegung auf Variablentypen nötig
  - keine Vorab-Reservierung von Speicher für Variablen
  - plattformunabhängig
- Bei Programmfehler: einfach fehlerhafte Zeile korrigieren und Programm neu starten
- sehr gutes Debugging: Entwicklungsumgebung springt direkt in fehlerhafte Zeile und zeigt, „von wo aus“ der fehlerhafte Code aufgerufen wurde
- je nach Interpreter-Art: keine Syntaxüberprüfung vor Programmlauf
  - Fehler in selten genutztem Code fallen evtl. erst sehr spät auf

Compiler/Interpreter Mischung

- Beispiel Java:
  - Compiler übersetzt Java-Programme in „Java Bytecode“
  - Interpreter (Java Virtual Machine, JVM) führt diesen Bytecode aus
  - vereint Vorteile aus Compiler- und Interpreter-Welt

siehe http://en.wikipedia.org/wiki/Java_bytecode

Was ist ein Projekt?

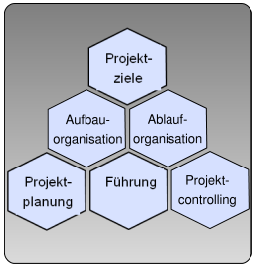
Ein Auszug aus den Definitionsvorschlägen …

- „Ein Projekt ist ein einmaliger Prozess, der aus einem Satz von abgestimmten und gelenkten Tätigkeiten mit Anfangs- und Endtermin besteht und durchgeführt wird, um unter Berücksichtigung von Zwängen bezüglich Zeit, Kosten und Ressourcen ein Ziel zu erreichen, das spezifische Anforderungen erfüllt.“[1]
- Definition nach DIN 69 901:
  „Ein Projekt ist ein Vorhaben, das im wesentlichen durch eine Einmaligkeit der Bedingungen in ihrer Gesamtheit gekennzeichnet ist.“
- Definition nach R. L. Marino (häufig zitiert)[2]:
  „A project is any task which has a definable beginning and a definable end and requires the expediture of one or more resources in each of the separate but interrelated and interdependent activities which must be completed to achieve the objectives for which the task was instituted.“

[1] Quelle: www.wikipedia.de, gelesen am 11.01.2009
[2] R.L.Martino: Project Management and Control, Vol.1, Finding the Critical Path, New York, 1964, p.17

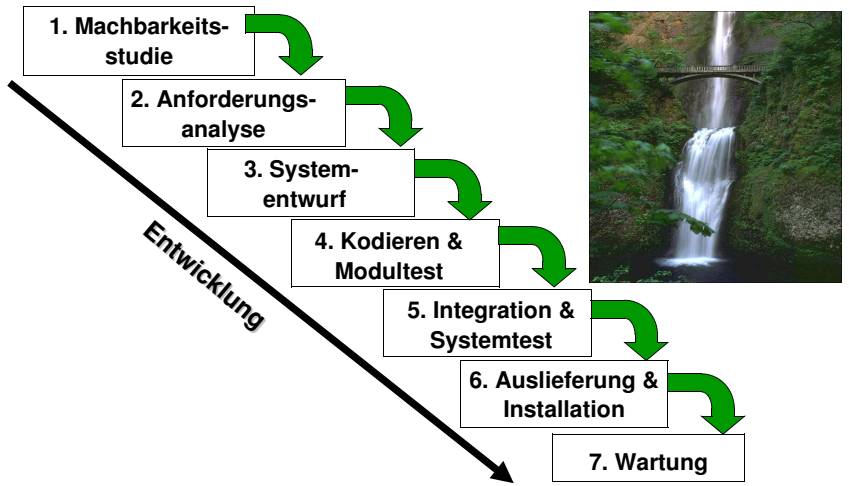
Was gehört zu einem Projekt?

- Eindeutige, abgrenzbare Zieldefinition
- Ressourcen
  - Personen, Material, Zeit, Geld
- Budget
- Zeitplanung
  - Meilensteine, Endtermin
- Organisation
  - Projektleiter als Zuständiger, Teilteams, Aufgabenzuordnung, …
  - Controlling, Qualitätsdefinition, QM-Plan
  - Meeting-Struktur, Meeting-Kultur
  - Verbindlichkeit



Ein Projekt wird i. a. als erfolgreich angesehen, wenn es abgeschlossen wird … „in time, in quality, in budget“

Projekt-Mgmt.: Wasserfallmodell



1. Machbarkeitsstudie

- Abschätzen des Gesamtprojekts bezüglich Kosten, Ertrag und Realisierung
- Problem informell und abstrahiert beschreiben, darauf aufbauend mögliche Lösungsansätze erarbeiten
- Ergebnis:
  - Lastenheft (grobe Anforderungsbeschreibungen)
  - Projektplan, Projektkostenkalkulation

Quelle für diese und die folgenden Folien:
http://cartoon.iguw.tuwien.ac.at:16080/fit/fit01/wasserfall/konzepte.html

2. Anforderungsanalyse

- Was genau soll die Software leisten?
- „Ist-Analyse“ durchführen, „Soll-Konzept“ (mit genauer Funktionalität der Software inkl. Systemeigenschaften wie Leistung, Schnittstellen, Portabilität) erstellen
- Anforderungen analysieren: Problem in Teilprobleme zerlegen; Lösungen finden und zu Gesamtlösung zusammensetzen
- Ergebnis:
  - Pflichtenheft, Analysedokument, Testfälle
  - 1. Version des Benutzerhandbuchs (noch keine Zeile Code geschrieben!)

3. Systementwurf

- Konkreter Entwurf: „Bauplan“ für die Software und die Software-Architektur
- Klare Definition der Schnittstellen
- Auswahl/Entwicklung erster Algorithmen
- Ergebnisse:
  - Entwurfsdokument mit Software-„Bauplan“
  - Testpläne für einzelne Komponenten

4. Kodieren & Modultest

- Anhand des Bauplans einzelne Komponenten kodieren (programmieren)
- einzelne Komponenten anhand der Testfälle überprüfen
- Ergebnis:
  - fertige Software-Komponenten
  - technische Dokumentation
  - Testprotokolle (der Komponenten)

5. Integration und Systemtest

- Zusammensetzen der einzelnen Komponenten zum Gesamtsystem
- Testen der Gesamtlösung
- Ergebnis:
  - fertiges vollständiges Software-Produkt
  - Benutzerhandbuch (endgültige Fassung)
  - Testprotokolle (für das Gesamtsystem)

6. Auslieferung & Installation

- Installation der Software beim Kunden (oder in-house)
- Einarbeitung/Schulung der Anwender
- ggf. vorher noch „Betaphase“ mit ausgewählten Benutzern zur Fehlersuche

7. Wartung

- Fehlermeldungen (Bug Reports) auswerten
- Verbesserungsvorschläge auswerten

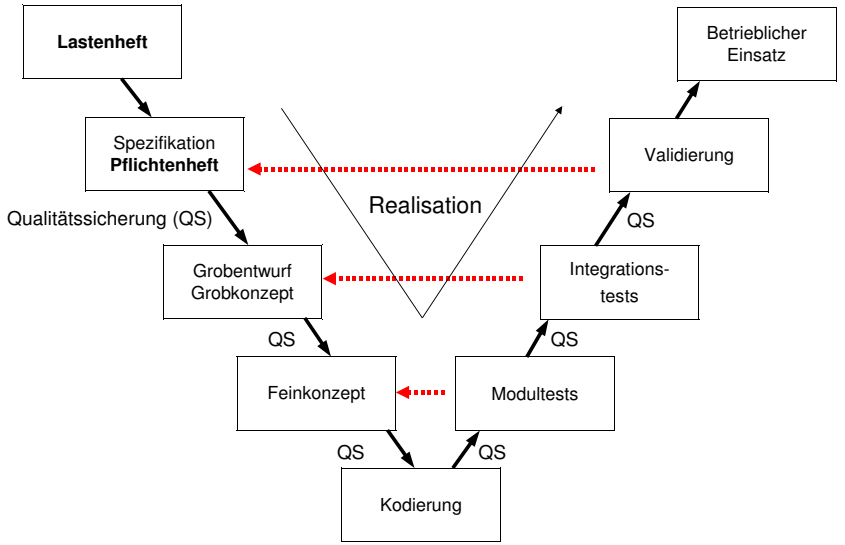
Vorteile Wasserfall-Modell

- Modell ist leicht verständlich und anwendbar,
- klare und einfache Strukturierung des Vorgehens,
- die Möglichkeit der direkten Abbildbarkeit des Modells auf eine Zeitachse,
- eine gute Basis für die weitere Definition eines Phasenmodells.

Nachteile Wasserfall-Modell

- anfangs nur ungenaue Ressourcenschätzungen möglich
- spätere Änderungen sind im Modell nicht vorgesehen
- In Entwurfsphase: Interaktion mit Endbenutzer mangels Prototypen sehr schwierig -> Benutzer bekommt erst dann richtige Vorstellung vom Produkt, wenn es fast fertig ist
- Pflichtenheft kann nie den Umgang mit dem fertigen System ersetzen -> Gutes Feedback ist erst sehr spät möglich
- größere Änderungswünsche nach Fertigstellung nur schwer und mit hohen Kosten erfüllbar

Projekt-Management.: V-Modell

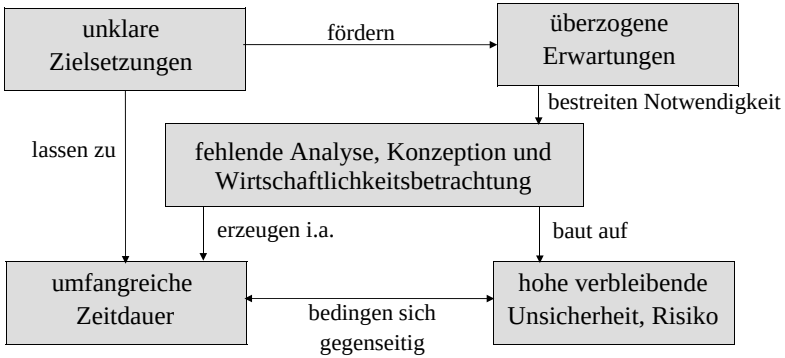


Warum scheitern Projekte?

- Projekterfolg = solution in time, in quality, in budget
- Projekte scheitern oft wegen mangelhafter und unzureichender
  - Menschenführung
  - Kommunikation
  - Planung
  - Ressourcen
  - Wissen
  - Selbstüberschätzung

Worauf ist zu achten?

- **Projektziele**
  - Eindeutigkeit, Abgrenzbarkeit
  - sagen, was man machen wird;
    sagen, was man *nicht* machen wird!



Spezifikation und Dokumentation

Eine **Spezifikation** ist eine

- **vollständige**,
- **detaillierte**,
- **unzweideutige**
- **Problembeschreibung**,
- die alle **Anforderungen** / gewünschten **Funktionen**,
- alle relevanten **Rahmenbedingungen**: alle **Hilfsmittel**, **Seiteneffekte** und **Konsequenzen** nennt und dabei
- klare Kriterien bzgl. der **Korrektheit und Übereinstimmung mit der Aufgabenstellung**
- sowie alle zu **erfüllenden Abnahmebedingungen** enthält

Lastenheft und Pflichtenheft

Ein Lastenheft definiert die Anforderungen der Anwender an die Leistungen eines Auftragnehmers

- Sollvorgaben an die spätere Funktionalität
- In der Theorie vom Auftraggeber/Fachabteilung erstellt
- siehe DIN 69905/VDI / VDE3694

Das Pflichtenheft ist die Gesamtleistungsbeschreibung (Spezifikation) der Funktionen, die durch eine IuK-Lösung erfüllt oder unterstützt werden sollen.

- Das Pflichtenheft enthält das Lastenheft und die vom Auftragnehmer vorgesehenen Realisierungsvorschläge.
- In der Theorie vom Auftragnehmer erstellt
- siehe DIN 69905/VDI / VDE3694