Die Phasen der Software-Entwicklung

2011-2018 — V1.8 [sw-entwicklung-phasen.txt]

1 Übersicht

Die Entwicklung (Development) von Software im Rahmen eines Projekts umfasst im wesentlichen die Phasen **Anforderungs-Definition** (_Requirement Spezification**),** Konzeption**,** Programmierung_ (Kodierung, Coding), **Debugging**, **Test**, **Dokumentation** und **Systemintegration**, die allerdings kaum jemals linear (Wasserfallmodell), sondern meist **zyklisch** (Spiralmodell) oder **iterativ** (Agiles Modell) durchlaufen werden. Bekannte **Methologien** zur Software-Entwicklung sind:

- Agile (Kanban, Scrum)
- Behaviour Driven Development (BDD)
- Cleanroom
- Lean
- Model Driven Development (MDD)
- Rapid Application Development (RAD)
- Rational Unified Process (RUP)
- Test Driven Development (TDD)
- V-Modell
- eXtreme Programming (XP)

Der **Entwicklungsaufwand** verteilt sich erfahrungsgemäß zu je 1/3 auf die **3 Hauptphasen**

- Anforderungs-Definition
- Konzeption und Kodierung
- Test und Systemintegration

2 Grundsätzliche Punkte

Die eigentliche Programmierung umfasst nur einen **kleinen Teil** des Gesamtaufwandes.

Man sollte sich von vorneherein auf das verwendete Modell der Software-Entwicklung einigen (selbst wenn es das **chaotische Modell** sein sollte!).

Die Festlegung eines **Projekt Managers** (freigestellt für das Projekt), eines **Projekt Management Tools** (zur Abschätzung der Aufwände, Zeiten und Kosten) sowie ein zentrales **Versionsverwaltungs- und Bugtracking-System** erleichtern Zusammenarbeit, Verwaltung, Reproduzierbarkeit, Fehlersuche, Dokumentation, Sicherung und Abrechnung des Projekts über den gesamten Konzeptions-, Entwicklungs- und Wartungszyklus hinweg.

An bestimmten Stellen im Projekt sollte ein **schriftliches Dokument** vorliegen, das von allen Beteiligten (Anwender, Auftraggeber, Projektleitung, Entwickler, Tester (QC), Dokumentationsabteilung, Geschäftsleitung, Marketing, \hdots) **akzeptiert** und **abgezeichnet** wird.

- **Mündliche Vereinbarungen** erfolgen zwar einfacher und schneller als schriftliche, allerdings wird dabei oft nicht genau überlegt ("das war doch nicht so gemeint"), sie geraten gerne in Vergessenheit und gehen nicht selten an der Projektleitung vorbei.
- **Schriftliche Vereinbarungen** dienen der Sicherheit aller Beteiligten, dass die Aufgabe korrekt verstanden und beschrieben wurde und adäquat gelöst werden kann. Mit ihrer Hilfe kann jederzeit ermittelt werden, wer wann welche Arbeiten in Auftrag gab und welcher Aufwand für sie anfiel.

Insbesondere sollte bei jedem erreichten **Meilenstein** (**Milestone**) eine **Abnahme** der Software durch den Auftraggeber erfolgen, deren Basis ein kontrollierter **Test** durch ausgewählte Anwender ist.

Ein **direkter Zugriff** von Anwendern auf Software-Entwickler sollte aus folgenden Gründen vermieden werden:

- Ständig Störungen.
- Ständiges "Nachschieben" von Funktionalitäten.
- Zwang zur schriftlichen Fixierung fehlt.
- Keine klare Kontrolle des Aufwandes möglich.

Der **Informationsaustausch** sollte statt dessen über die Projektleitung bzw. die **QC** (Qualitätskontrolle) auf Basis eines Tools zur Request-Erfassung **kanalisiert** werden:

Die einvernehmliche Festlegung der **Abrechnungsmodalitäten** und der Zeitpunkte der Abrechnung (Erreichen bestimmter Meilensteine) ist notwendig. Typische wesentliche (und aufwendige) Schritte, die für sich bereits einen Wert darstellen und daher einzeln abgerechnet werden sollten — zumindest falls die Entwicklung danach nicht in Auftrag gegeben wird — sind:

- Pflichtenheft-Erstellung
- Aufwand-Schätzung
- Prototyp-Erstellung

Um die **Wartung** (Maintenance) zu erleichtern, sollten Kommentare (Programmkopf), Namenskonventionen, Formatierungsstil und Art und Umfang der Programm-Dokumentation festgelegt und durch Tools automatisch geprüft werden (z.B. beim Einchecken in die Versionsverwaltung):

- Erhöht die Verständlichkeit.
- Verringert die Fehleranfälligkeit.
- Macht dem Wartungs-Programmierer das Leben leichter.

Steckt das **Wissen über eine Software** nur in den Köpfen der Entwickler, so reduziert es sich mit einer Halbwertszeit von etwa 3-6 Monaten aufgrund der Übernahme neuer Projekte und der natürlichen Fluktuation der Entwickler. Die **Entwickler- und die Anwender-Dokumentation** sind daher unbedingt erforderlich, falls eine Software langfristig eingesetzt werden soll.

Nach einer gewissen Laufzeit des erstellten Systems (z.B. 2-3 Jahre) ist über ein **Folgesystem** nachzudenken. Neue Hardware, Programmiersprachen, Datenbanken, Tools und Methoden (z.B. Funktionale Programmierung) etablieren sich auf dem Markt und verdrängen die bisher verwendeten. Evtl. sind auch keine Entwickler mit Erfahrung in den bisher eingesetzten Tools zu finden. Dann beginnt der ganze Prozess von vorne.

3 Die 3 Hauptphasen

Die 3 Hauptphasen umfassen folgende Tätigkeiten:

- **Anforderungs-Definition**
  - ▷ Ggfs. Analyse des Altsystems + Sammeln von Mängeln/Erweiterungswünschen
  - ▷ Anforderung der Anwender ⇒ **Lastenheft**
  - ▷ Analyse durch Entwickler ⇒ **Pflichtenheft** + Aufwandschätzung
    - ○ **Meilensteine** festlegen (definierte Entwicklungsstände mit Abnahme)
  - ▷ Evtl. **Prototyp** der GUI-Oberfläche erstellen und mit Anwender diskutieren
  - ▷ Pflichtenheft (und Prototyp) **schriftlich** vom Anwender absegnen lassen
  - ▷ Auftrag **schriftlich** vom Anwender erteilen lassen
- **Konzeption und Entwicklung**
  - ▷ **Entwurf** (Design)

        - Grundstruktur (**Architektur**) der Software (Use Cases, UML)
        - Objekte/Subjekte (nur problemrelevante, OOA/OOD)
            - Eigenschaften/Attribute (nur problemrelevante)
            - Relationen/Beziehungen (nur problemrelevante)
        - Schnittstellen-Identifikation (**Interfaces**)
            - Interne (zw. Objekten/Subjekten)
            - Externe (nach außen)
        - Algorithmen
        - Datenbank-Tabellen und -Beziehungen (ORM = Objekt-relationales Mapping)
        - Design der GUI-Oberfläche (**Usability**)
    - **Implementierung** ("eigentliche" Codierung)
        - Schrittweise
        - Zerlegung in Teilaufgaben
        - Modularisierung + Schnittstellen
        - Versionsverwaltung
        - Tests definieren + implementieren (umfangreiche **Testdaten** erstellen)
        - Ständige Integrationstests (täglich oder wöchentlich)
    - **Alpha-Test** durch Entwickler selbst (möglichst automatisiert!)
    - **Programm-Dokumentation** (während Entwicklung!, möglichst im Quellcode!)
    - Release-Management
    - Freigabe für die Qualitätskontrolle (QC)
- **Test, Dokumentation und Systemintegration**
    - **Beta-Test** durch die QC (Qualitätskontrolle; Wichtig: keine Entwickler!)
        - Debugging und Fehlerbehebung durch Entwickler ("Showstopper")
    - Integrationstest (in Produktionsumgebung)
    - Freigabe für Anwendertest
    - **Gamma-Test** durch Anwender (Massentest) $\Rightarrow$ **schriftliche** Abnahme
    - **Anwender-Dokumentation** (evtl. nicht notwendig; oft als zu teuer angesehen)
    - Freigabe an die Produktion
    - Übergabe an Wartungs-Programmierer

4 Freigabe und Wartung

Die abschließende **Freigabe** (Release) für den Einsatz in der Produktion und die ständige **Wartung** (Maintenance) der Software sind zwei wesentliche Schritte, die bei der Aufwandabschätzung gerne vergessen werden, obwohl sie wesentliche Zeit- und Kostenfaktoren darstellen und für sich bereits große Probleme darstellen können.

- **Produktionsfreigabe** (**Deployment**)

    - ▷ Konfiguration (**Configuration Management**): Anpassung an Zielplattfoßm
    - ▷ Installation auf Produktionsrechner
        - ○ Datentransfer der Altdaten (Konvertierung)
    - ▷ Produktionstest
        - ○ Parallellauf Alt+Neusystem
        - ○ Nachweis der Korrektheit (Erklärung von Abweichungen)
        - ○ Ablösung Altsystem
    - ▷ Anwenderschulung
- **Wartung**/**Pflege** (**Maintenance**)
    - ▷ Softwaresystem überwachen
    - ▷ Anwendungsdaten sichern / zurückspielen
    - ▷ Benutzerfahrungen sammeln (**User Experience**)
    - ▷ Von Anwender berichtete Fehler (**Bugs**) aufzeichnen (**Ticket Request**) und reproduzieren (**Test schreiben**)
        - ○ Kategorisierung der Fehler
            - ∗ leicht = ignorieren
            - ∗ mittel = nächster Release (Planung)
            - ∗ schwer = sofort (Patch/Workaround)
        - ○ Fehlersuche und Fehlerbehebung durchführen
        - ○ Patch/Workaround
            - ∗ Erstellen
            - ∗ Testen
            - ∗ Einspielen
            - ∗ Dokumentieren (Doku erweitern)
            - ∗ Anwender nachschulen
            - ∗ Versionsverwaltung
    - ▷ Qualitätssicherung (**Quality Assurance**, QA)
    - ▷ **Refactoring** (Umorganisation/Restrukturierung des **Legacy**/Altcodes)
    - ▷ Übergabe an neue Mitarbeiter

5 Abschliessende Bemerkungen

Programme werden wesentlich öfter gelesen als geschrieben! Man sollte also immer so programmieren, als würde man für "einen anderen" schreiben (z.B. den Wartungs-Programmierer oder sich selbst nach ein paar Monaten) und den Quellcode so übersichtlich und lesbar wie möglich schreiben und formatieren.

Ohne Wartung sind Software-Produkte langfristig gesehen zum Tode verurteilt! Die Wartung verursacht über die Laufzeit einer Software gesehen häufig viel höhere Kosten als die ursprüngliche Entwicklung der Software.

Eine Organisation sollte fähig sein, aus ihren Fehlern zu lernen und ihre internen Abläufe permanent zu verbessern! Dies gilt ebenso für die Software-Entwicklung. D.h. der gesamte Prozess der Software-Entwicklung sollte den gleichen Methoden der Überwachung und permanenten Optimierung unterworfen sein wie jede andere Aufgabe innerhalb einer Firma auch.

Eine vernünftige Fehlerkultur verteufelt den Fehler bzw. den Verursacher des Fehlers nicht. Fehler kann man nur erkennen und beheben, wenn sie bekannt gemacht und aufgezeichnet werden. Eine Organisation, die das Zugeben von Fehlern erschwert oder bestraft, wird scheitern (weil die Fehler aus Angst verheimlicht werden und man somit nichts daraus lernen kann).

— END —