Projektmethoden, Tools

Kenntnisse über Softwareprozessmodelle

Ein Software-Prozessmodell ist ein Modell für den Ablauf der Entwicklung eines Software-Systems. Dabei geht es nicht um die Darstellung des Ablaufs eines bestimmten Software-Entwicklungsprojekts, sondern einer ganzen Klasse von Projekten. Dazu wird der Entwicklungsprozess üblicherweise in Phasen eingeteilt.

Man unterscheidet typischerweise zwischen

Planung des Prozesses.
Spezifikation der Anforderungen an das Produkt.
Design des Software-Produkts.
Implementierung (Kodierung) und diversen Tests des Software-Produkts.

Im Rahmen eines Prozessmodells werden neben den Aktivitäten innerhalb dieser Phasen auch die Rollen und Qualifikationen der Mitarbeiter definiert, die gewisse Aktivitäten durchführen oder für sie verantwortlich sind. Außerdem werden die Dokumente und Unterlagen, zusammenfassend Artefakte1 genannt, festgelegt, die im Rahmen des Entwicklungsprozesses erstellt werden müssen. Oftmals sind hier-bei auch Vorgaben von Behörden zu beachten. Es gibt nicht das Prozessmodell für alle Typen von Software-Entwicklungsmodellen. Ausschlaggebend für die Wahl eines Prozessmodells ist der Projekttyp.

Kenntnisse über den Aufbau des Wasserfallmodells

Das Wasserfallmodell (englisch: waterfall model) ist ein lineares Vorgehensmodell, das Entwicklungsprozesse in aufeinanderfolgende Projektphasen unterteilt. Im Gegensatz zu iterativen Modellen wird jede Phase nur einmal durchlaufen. Die Ergebnisse einer jeden Vorgängerphase gehen als Vorannahmen in die Folgephase ein. Zur Anwendung kommt das Wasserfallmodell insbesondere in der Software-Entwicklung.

Die Entwicklung des klassischen Wasserfallmodells wird dem Computerwissenschaftler Winston W. Royce zugeschrieben. Als Alternative stellt Royce ein iterativ-inkrementelles Modell vor, bei dem jede Phase auf die vorhergehende zurückgreift und deren Ergebnisse verifiziert.

Ein Modell mit sieben Phasen, dass in mehreren Durchgängen (Iterationen) durchlaufen wird:

- Systemanforderungen
- Software-Anforderungen
- Analyse
- Design
- Implementierung
- Test
- Betrieb

Probleme, die beim Wasserfallmodell auftreten können

Vorteile:

- Einfache Struktur durch klar abgegrenzte Projektphasen.
- Gute Dokumentation des Entwicklungsprozesses durch klar definierte Meilensteine.
- Kosten und Arbeitsaufwand lassen sich bereits bei Projektbeginn abschätzen.
- Projekte, die nach dem Wasserfallmodell strukturiert werden, lassen sich auf der Zeitachse gut abbilden.

Nachteile:

- Komplexe oder mehrschichtige Projekte lassen sich nur selten in klar abgegrenzte Projektphasen unterteilen.
- Geringer Spielraum für Anpassungen des Projektablaufs aufgrund veränderter Anforderungen.
- Der Endanwender wird erst nach der Programmierung in den Produktionsprozess eingebunden.
- Fehler werden mitunter erst am Ende des Entwicklungsprozesses erkannt.

Kenntnisse über den Aufbau des V-Modells

Das V-Modell ist ein Modell, das für verschiedene Entwicklungsprozesse genutzt wird, z. B. in der Softwareentwicklung. Es wurde in seiner ursprünglichen Form in den 1990er Jahren entwickelt, im Laufe der Zeit immer weiter verfeinert und an moderne Entwicklungsmethoden angepasst. Die grundlegende Idee stammt allerdings schon aus den 1970er Jahren und war als eine Art Weiterentwicklung des Wasserfallmodells konzipiert.

Zusätzlich zu den jeweiligen Entwicklungsphasen eines Projekts definiert das V-Modell parallel die begleitenden Vorgehensweisen zur Qualitätssicherung und beschreibt, wie diese einzelnen Phasen miteinander interagieren können. Seinen Namen verdankt das Vorgehensmodell seiner dem Buchstaben V ähnelnden Struktur.

Die einzelnen Phasen des V-Modells:

Zunächst definiert das V-Modell den Ablauf eines Projekts in einzelnen Phasen, die immer weiter ins Detail gehen:

- Zu Beginn des Projekts sieht das Modell eine Analyse der allgemeinen Anforderungen an das geplante System vor.
- Danach soll das Projekt mit funktionalen und nichtfunktionalen Anforderungen an die Systemarchitektur angereichert werden.
- Darauf folgt der Systementwurf, bei dem die Komponenten und Schnittstellen des Systems geplant werden.

- Sind diese Phasen abgeschlossen, kann die Softwarearchitektur im Detail konzipiert werden.

Nun folgt die eigentliche Entwicklung der Software gemäß diesen Plänen. Anschließend folgen die Phasen der Qualitätssicherung, die sich immer auf die Schritte der Entwicklung beziehen.
Das Modell sieht folgende Aufgaben vor:

- Unit Tests
- Integrationstests
- Systemintegration
- Abnahme

Kenntnisse über Vor- und Nachteile des V-Modells

Die Vorteile des V-Modells:

- Optimierung der Kommunikation zwischen den Beteiligten durch fest definierte Begriffe und Zuständigkeiten
- Minimierung von Risiken und bessere Planbarkeit durch fest vorgegebene Rollen, Strukturen und Ergebnisse
- Verbesserung der Produktqualität durch fest integrierte Maßnahmen zur Qualitätssicherung
- Kosteneinsparung durch transparente Aufarbeitung des gesamten Produktlebenszyklus

Insgesamt kann das Modell dabei helfen, Missverständnisse und unnötige Arbeiten zu vermeiden. Außerdem sorgt es dafür, dass alle Aufgaben zum richtigen Zeitpunkt und in sinnvoller Reihenfolge erledigt werden und dass möglichst keine Leerlaufzeiten entstehen.

Die Nachteile des V-Modells:

Das Vorgehensmodell ist teils zu simpel, um den Entwicklungsprozess aus Sicht der Entwickler vollständig abzubilden. Der Fokus liegt verstärkt auf dem Projektmanagement. Zudem erlaubt die starre Struktur kaum, flexibel auf Änderungen während der Entwicklung zu reagieren, und fördert somit einen vergleichsweise linearen Projektverlauf. Dennoch ist es möglich, mit dem V-Modell agile Entwicklung zu betreiben, wenn das Modell richtig verstanden und genutzt wird.

Kenntnisse über Agiles Projektmanagement

Agiles Projektmanagement:

bezeichnet Vorgehensweisen, bei denen das Projektteam über hohe Toleranzen bezüglich Qualität, Umfang, Zeit und Kosten verfügt und eine sehr hohe Mitwirkung des Auftraggebers bei der Erstellung des Werks erforderlich ist. Charakteristisch für Agiles Projektmanagement ist die Fokussierung auf das zu liefernde Werk und die Akzeptanz durch die Anwender. Hingegen werden geschäftliche Anforderungen, wie z.B. die Termintreue, Kostentreue oder Erfüllung eines spezifizierten Leistungsumfangs weniger oder nicht berücksichtigt.

Fachbegriff Scrum Master

Scrum Master: Mediator für agile Teams.

Scrum, ein Modell des agilen Produktmanagements, kennt drei Rollen:

- Neben dem eigentlichen Entwicklungsteam und dem Product Owner, der für Qualität und Funktionsumfang der Produkte zuständig ist, gibt es noch den Scrum Master.
- Auch wenn er nicht direkt an der Entwicklung eines Produkts beteiligt ist, nimmt er einen großen Stellenwert im Gefüge von Scrum ein.
- Ein Scrum Master ist Trainer, Mediator, Moderator und Assistent gleichermaßen.

Das Modell Scrum war ursprünglich nur für die Entwicklung von Software gedacht. Um Entwicklerteams dynamischer und effizienter zu gestalten, hat man diese agile Methode entwickelt und konkrete Regeln (das Framework) etabliert. Die Abläufe lassen sich aber auch genauso gut auf andere Teams und Produkte anwenden. Egal, ob Software, Hardware oder Dienstleistungen entwickelt werden – der Scrum Master muss immer Teil des Geschehens sein.

Der Scrum Master und seine Aufgaben:

- Installation von Scrum
- Laufendes Coaching
- Überwachung der Scrum-Prozesse
- Überprüfung der Scrum-Artefakte
- Moderation & Organisation von Meetings
- Kommunikation & Mediation
- Teambuilding
- Beseitigung von Hindernissen
- Schutz des Teams

Fachbegriff Productowner

Als Product Owner (deutsch: Produkteigentümer) bezeichnet man eine der sechs Rollen im Scrum Prozess. Der Product Owner gehört dabei neben dem Scrum Master und dem Entwicklungsteam zu den zentralen Akteuren. Diese Person ist für die Wertsteigerung des Produkts im Entwicklungsprozess verantwortlich und leitet das Team an.

Aufgaben des Product Owners:

Der Product Owner nimmt im Scrum Team eine Vielzahl von Aufgaben wahr. Er kommuniziert regelmäßig mit dem Kunden. Er fängt dessen Wünsche und Bedürfnisse ein und vertritt diese gegenüber dem eigenen Team. Der Product Owner ist dafür verantwortlich, dass die Vorgaben des Kunden in der Softwareentwicklung berücksichtigt werden.

Weitere Aufgaben des Product Owners sind:

Produkteigenschaften:
Der Product Owner bestimmt, welche Eigenschaften das Endprodukt in Abhängigkeit von den Wünschen des Kunden und der Nutzer aufweisen soll.

Priorisierung:
Er ist dafür zuständig, die Prioritäten für die Abarbeitung einzelner Arbeitspakete festzulegen.

Kontrolle:
Er begutachtet die Funktionalität zum Ende eines Sprints und stellt fest, ob diese dem Kunden präsentiert werden kann oder noch nicht akzeptabel ist.

Pflege des Backlogs:
Der Product Owner pflegt das Backlog. Hierfür entwickelt er User Stories, aus denen sich die Aufgaben für das Entwicklerteam ableiten lassen.

Aufgabenzuweisung:
Er bestimmt, welche Aufgaben das Entwicklerteam als nächstes abarbeiten soll. Wie das Team diese Aufgaben erledigt, fällt allerdings nicht in seinen Zuständigkeitsbereich. Sind die Aufgaben erst einmal erteilt, bleiben sie bestehen und werden nicht während eines Sprints willkürlich wieder geändert.

Verantwortlichkeit:
Der Product Owner ist für den Erfolg oder Misserfolg des Projekts verantwortlich.

Der Product Owner ist niemals ein direkter Bestandteil des Entwicklungsteams. Er kann also nicht als Entwickler oder ähnlicher Spezialist aktiv mitarbeiten, da sich aus diesen zwei mehr oder weniger gegensätzlichen Positionen Interessenkonflikte ergeben könnten.

Fachbegriff Backlog

Der englische Begriff Backlog heißt ins Deutsche übersetzt "Auftragsbestand" bzw. "Arbeitsrückstand" und bezeichnet allgemein eine dynamische Liste an gesammelten Aufgaben, Features oder Anforderungen, die auf ihre Abarbeitung warten.

Im Projektmanagement beschreibt der Begriff Backlog allgemein ausgedrückt alle projektbezogenen Aufgaben, die noch zu erledigen sind. Im agilen Projektmanagement wird unterschieden zwischen einem Product Backlog und einem Sprint Backlog.

Der Begriff Product Backlog bezeichnet in der Softwareentwicklung alle gesammelten Anforderungen an die zu entwickelnde Software und wird in der Regel im Zusammenhang mit dem agilen Vorgehensmodell Scrum verwendet.

Jede zu entwickelnder Software besitzt genau einen Product Backlog und einen Product Owner. Wichtig ist, dass ein Product Backlog niemals vollständig ist, da er vor allem zu Beginn der Entwicklung nur Anforderungen beinhaltet, die am besten verstanden wurden. Neben den Anforderungen, umfasst ein Product Backlog auch Bugs, die gefixt werden müssen, oder nötige Verbesserungen in Form von User-Stories, Epics oder Tasks. Alle Backlog-Items sollten eine genaue Beschreibung, eine Priorität und eine Aufwandschätzung vorweisen.

Die Priorisierung der sogenannten Backlog-Items, erfolgt durch den Product Owner - häufig auch in Zusammenarbeit mit dem Entwicklerteam. Wie bereits erwähnt, ist der Backlog niemals vollständig und muss vom Product-Owner regelmäßig gepflegt werden, da bestehende Backlog-Items durch neue Anforderungen häufig neu priorisiert werden müssen oder dadurch neue Aufgaben hinzukommen. Gleichzeitig werden die Beschreibungen der einzelnen Anforderungen angepasst, da Backlog-Items mit höheren Prioritäten idealerweise eine noch detailliertere und feinere Beschreibung haben als die mit niedriger Priorität.

Wichtig ist also, dass es sich hierbei um eine dynamische Liste handelt, welches verändert und angepasst wird und somit nicht zu verwechseln ist mit einem Lasten- oder Pflichtenheft.

Fachbegriff Sprint

Sprint Backlog:
Im Sprint Backlog befinden sich Projektaufgaben, die aus dem Product Backlog ausgewählt werden und im kommenden Sprint umgesetzt werden sollen, um das Sprint-Ziel zu erreichen. Alle Aufgaben sind mit Zuständen versehen, sodass erledigte Aufgaben auch als solche markiert werden. Somit wird das Team befähigt, jederzeit den Fortschritt der Entwicklung zu erkennen und gleichzeitig den Überblick über den Gesamtfortschritt zu behalten. Die einzelnen Items werden vom Entwicklungsteam realisiert und aktualisiert.

Fachbegriff Stakeholder

Stakeholder haben ein Interesse am Ergebnis Ihres Projekts. Für gewöhnlich handelt es sich dabei um die Mitglieder eines Projektteams, Projektmanager, Führungskräfte, Projektsponsoren, Kunden und Nutzer. Stakeholder sind Personen, die an dem Projekt beteiligt sind und zu irgendeinem Zeitpunkt von Ihrem Projekt betroffen sein werden. Ihre Beiträge können das Ergebnis direkt beeinflussen. Es empfiehlt sich, auf ein gutes Stakeholder-Management zu achten und zur Zusammenarbeit an dem Projekt kontinuierlich mit Stakeholdern zu kommunizieren. Immerhin haben sie ein Interesse am Ergebnis.

Denn egal wie reibungslos ein Projekt intern ablaufen mag:
Ohne die Berücksichtigung der Stakeholder kann es gut sein, dass man

- am Ziel vorbei arbeitet.
- mögliche Hindernisse und Widerstände nicht erkennt.
- wichtige Betroffene nicht mit ins Boot holen.
- mögliche Chancen nicht sieht.
- auf ein Scheitern des Projektes zusteuert.

Fachbegriff Daily Scrum/Daily Standup

Das Daily Scrum ist ein kurzes, vom Scrum Master moderiertes Meeting für die Kommunikation innerhalb des Teams und zur Feststellung des aktuellen Projektstatus. Die Teammitglieder informieren sich gegenseitig über ihren Fortschritt sowie eventuelle Probleme und gehen Commitments für die laufende Iteration oder den laufenden Sprint ein. Das Daily Scrum ist eine fokussierte Unterhaltung mit einem strengen Zeitrahmen und wird jeden Tag zur gleichen Zeit (idealerweise am Morgen) am gleichen Ort abgehalten. Das Scrum Task Board steht dabei im Mittelpunkt.

Nutzung des Daily Standup:

Die Teammitglieder beantworten dabei nacheinander die folgenden drei Fragen:

- Woran habe ich gestern gearbeitet?
- An was arbeite ich heute bzw. was werde ich heute fertigbekommen?
- Was hindert mich daran, meine Commitments einzuhalten?

Die gesamte Konversation im Daily Standup sollte sich um diese drei Fragen drehen. Alle weiterführenden Diskussionen, die daraus hervorgehen, sollten separat geführt werden.

Nur diejenigen, die auch in den jeweiligen Sprint involviert sind, sollten an dem Daily Scrum teilnehmen.

Zudem sollte bei den drei Fragen folgendes berücksichtigt werden:

- Die oben genannten Punkte sollten kurz behandelt und formuliert werden.
- Jeder Teilnehmer hat einen eigenen Zeitrahmen, um seine Punkte zu erklären.
- Es geht nicht um die Beurteilung dieser Tätigkeiten, sondern um die Transparenz.
- Mögliche Action Items können daraus entstehen (vor allem für den Scrum Master, wenn es um Impediments geht)
- Alle Gespräche innerhalb des Daily Standup sollten sich auf die drei genannten Fragen beziehen.

Fachbegriff User Story/Story Board

User Storys:

sind Nutzergeschichten und Grundlage für ein Scrum-Projekt. Der Product Owner beschreibt in einer User Story die Anforderungen aus Nutzersicht: Wie wünschen sich Anwender eine Software oder Kunden einen Liefergegenstand, um ihre Geschäftsziele zu erreichen.

Obwohl der Begriff „User Story" im Scrum Guide nicht auftaucht, nutzen agile Arbeitstechniken ihn häufig. User Storys stehen im Backlog neben Epics und Tasks, Qualitätsanforderungen, Fehlern und Verbesserungen. Eine User Story besteht aus wenigen, leicht verständlichen Sätzen. Sie sind kurz, aus Kundensicht jedoch spezifisch und detailliert.

User Storys transportieren nicht nur Erwartungen an zukünftige IT-Systeme, sondern auch Anforderungen an Liefergegenstände, die agile Projekte realisieren. User Storys bewegen sich im Anforderungsraum, ohne in den Lösungsraum zu wechseln. Sie geben keine technischen Lösungen vor, die die Entwickler zum sturen Realisieren zwingen.

Beispiele für User Storys:

Folgende User Storys zeigen, worum es jeweils gehen kann und wie einfach eine User Story im ersten Schritt formuliert sein kann:

- Ein neuer Kunde soll sich bei einem E-Learning-Portal registrieren, um sich auf eine Zertifizierung vorzubereiten.
- Ein Kunde möchte Waren in einem Webshop auswählen und dann bestellen.
- Ein IT-Administrator möchte Datenbanken verwalten, indem er Datensätze anlegt, ändert oder löscht.
- Eine Abteilung in einem internationalen Konzern soll in einen neuen Bereich und damit in eine neue Struktur überführt werden.

Story Board:

Im Gegensatz zum Drehbuch, das der konzeptuellen Inhaltsrepräsentation dient, wird ein Storyboard als visuelle Vorlage für die Erstellung von Bildinhalten genutzt. Es stellt Handlungsverläufe bildlich dar, ist stark ablauforientiert und vermittelt so einen ersten Eindruck für die spätere Umsetzung.

Was gehört alles in ein Story Board:

Ein Storyboard ist eine Bildsequenz in Kombination mit einem Text, die die Einstellungen eines Filmes (hier: Einstellung der Kamera), einer Multimedia-Produktion oder anderer Formate der darstellenden Kunst visualisiert.

Wie macht man ein Story Board:
Wenn Sie ein Storyboard erstellen wollen, beginnen Sie mit einem Skript. Selbst ein sehr einfaches Skript zeigt die Richtung auf, die Sie für Ihr Video geplant haben. Es hilft Ihnen, Ihre Geschichte zu erzählen. Teilen Sie Ihr Skript in Szenen auf und segmentieren Sie die Szenen dann in einzelne Aufnahmen.

Fachbegriff Softwareentwurf

In der Entwicklung dient der Software-Entwurf als technischer Plan für eine zu entwickelnde Software, um die Anforderungsanalyse technisch umzusetzen. Der Software-Entwurf kann maßgeblich dabei helfen, die Entwicklung zu rationalisieren.

Bei der gezielten Entwicklung eines Systems oder einer Software müssen unterschiedliche Projektphasen berücksichtigt werden, um unnötige Korrekturen zu minimieren. Fußend auf der Anforderungsanalyse und -definition erfolgt mit dem Software-Entwurf eine Projektphase, die sich gezielt mit der Übersetzung der Anforderung in verschiedene Systemkomponenten beschäftigt.

Diese Planungsphase erfolgt also mit IT-Fachpersonal und überträgt die Anforderungen der Kundenseite in strukturelle Eigenschaften und Komponenten. Es handelt sich hierbei um einen Lageplan, anhand dessen gezielter programmiert werden kann und der das Gerüst des Codes angibt, mit dessen Hilfe die Anforderungsanalyse erfüllt werden soll.

Wie hilft der Software-Entwurf beim Programmieren:
Ein Software-Entwurf (der einfacheren Lesbarkeit halber hier mit Bindestrich, im Deutschen sonst auch oft Softwareentwurf) ist ein Hilfsmittel für Programmiererinnen und Programmierer. Dabei wird fachspezifisches Wissen angewandt, um einen Grundriss für die Projektarbeit anzugeben.

Ein Software-Entwurf besteht dabei aus verschiedenen Teilen, die genau definieren, wie die Programmierung letztlich erfolgt:

Programmiersprache:
Im Entwurf wird festgelegt, in welcher Sprache programmiert wird. Hierbei spielen die Kompetenzen der beteiligten Programmiererinnen und Programmierer ebenso eine Rolle wie die Wünsche der Beauftragenden, das Pflichten- und Lastenheft und die Kompatibilität der Software. In einem Software-Entwurf wird gegebenenfalls auch zwischen unterschiedlichen Programmiersprachen abgewogen.

Entwicklungsumgebung:
In welcher Integrated Development Environment soll der Code geschrieben und kompiliert werden? Nutzt das Team stets die gleiche IDE oder bietet sich eine andere Umgebung für die spezifischen Projektanforderungen an?

Library:
Der Software-Entwurf klärt ebenfalls, ob bestimmte Code-Bausteine aus einer Library für die Software genutzt werden können und sollen oder ob der Quellcode komplett neu geschrieben werden muss. In dieser Phase gibt es eine Schnittmenge zum Projektmanagement, da Libraries die Arbeitszeit enorm verkürzen können.

Application Programming Interfaces:
Auch die Frage nach den APIs und der Integration von Code und Front End wird im Software-Entwurf besprochen. Fragen nach Library, Codesprache und API sollten stets in

Abhängigkeit zueinander definiert werden und stets die gestellte Anforderungsanalyse berücksichtigen.

Software Development Kits:

Handelt es sich um Projekte, die in einer festen Umgebung arbeiten sollen, so können auch SDKs genutzt werden. Beispiele hierfür sind Android Studio oder Windows .NET – zwar limitiert dies die Flexibilität der Software, doch erleichtert den Workflow für alle beteiligten Developer. Je nach Anforderungsanalyse kann die Nutzung eines SDK also angebracht sein.

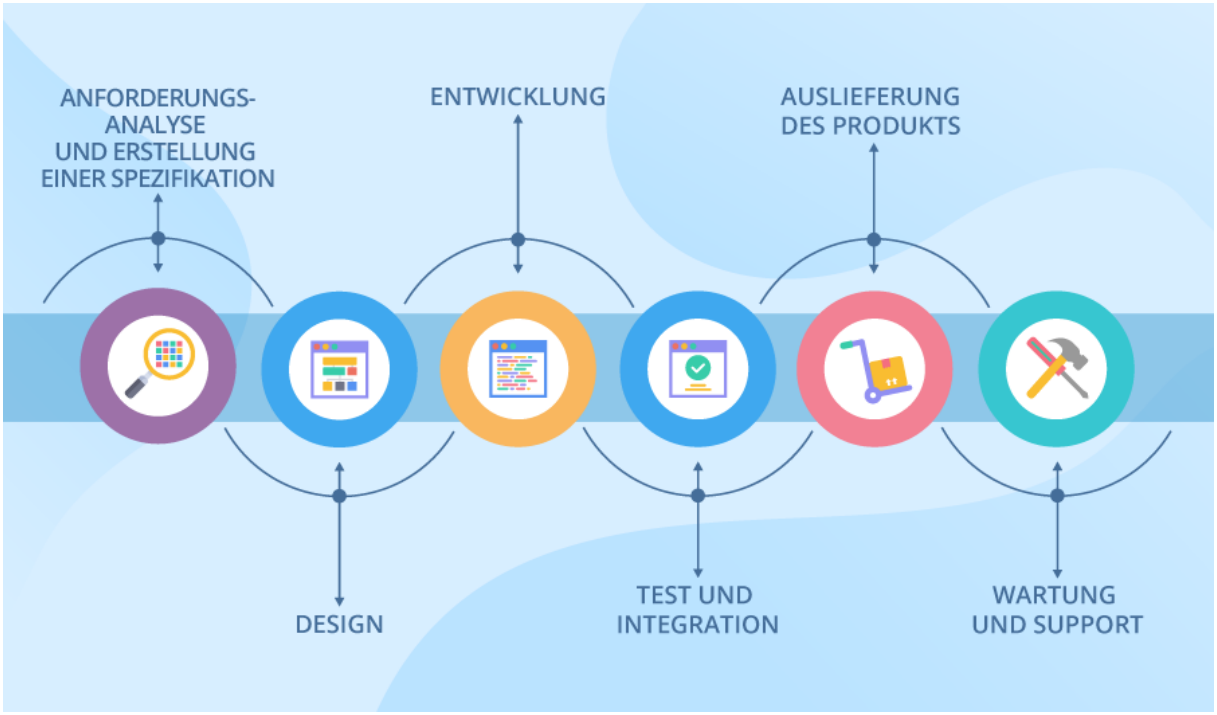
Woraus besteht ein Software-Entwurf:

Zwar ist ein Software-Entwurf in der Summe ein vollständiger Plan der späteren Software, beschreibt jedoch eher die Breite der Software, ohne dabei in die Tiefe zu gehen. Im Entwurf werden alle Systeme und Bausteine (Module, Klassen, Komponenten) beschrieben und die Beziehungen zueinander definiert. Zudem werden die Schnittstellen der Architekturbausteine beschrieben.

Im Ergebnis steht ein möglichst spezifischer Überblick darüber, wie die Anforderungsanalyse technisch erfüllt werden soll, sowie ein kompletter Abriss über das gesamte System. Dass dennoch zu diesem Zeitpunkt keine einzige Zeile Code geschrieben wurde, erleichtert das spätere Programmieren – Änderungen können in der Entwurfsphase noch sehr leicht vorgenommen werden, etwaige Probleme in Umgebungen oder unzureichende Libraries werden hier schnell erkannt und es kann gegengesteuert werden.

Stadien der Softwareentwicklung

Ausgehend von der Definition des Projektes, ist das hier vorgestellte Modell eine sequenzielle Darstellung der einzelnen Phasen, also von der ersten Idee bis zum produktiven Einsatz der Software.



Phase 1: Voruntersuchung

Sie dient zur Auftragsklärung und hat das Ziel die Anforderungen (Requirements) für die Realisierung des Projektes zu definieren und eine erste Zeit- & Aufwandsschätzung zu machen.

Für die Voruntersuchung wird der Ist-Zustand aufgenommen:

Die zu analysierenden Aufgaben werden definiert, und die Ziele dieser Analyse werden festgelegt. Mit Hilfe eines aufgestellten Fragenkatalogs wird die Analyse durchgeführt, die Ergebnisse ausgewertet und die Anforderungen definiert, die der Auftraggeber prüft und freigibt.

Danach folgt die Machbarkeitsstudie:

Die grundsätzliche Lösungsalternativen werden beschrieben und bewertet, eine Zeit- und Aufwandsschätzung wird erstellt und ein Durchführungsplan generiert.

Phase 2: Konzeption/Grobkonzept

Klärung was das IT-System leisten soll: Funktionalität, Datenhaushalt usw. Der Kunde ist gefordert ein detailliertes Lastenheft zu liefern.
Zum Schluss dieser Phase soll der Vertrag zwischen Aufraggeber und Auftragnehmer stehen: nach Vorlage und Abnahme des Pflichtenheftes (beinhaltet u. a. das Sollkonzept, den Durchführungsplan und eine formalisierte Systemspezifikation) und Akzeptanz des Angebotes.
Erst dann kann das Sollkonzept dargestellt und die benötigte Hardware- und Software geklärt und fixiert werden.

Phase 3: Design/System und Komponentenentwurf

In dieser Phase wird geklärt, wie das IT-System aufgebaut wird. Das heißt dass die IT-Fachleute die Hardware und Softwarearchitektur klären müssen, die Komponenten (einzelne Baugruppen innerhalb der Softwarearchitektur) spezifizieren und die Integration der Komponenten in die geplante IT-Landschaft planen. Am Ende stehen die Architektur und der Projektplan für die Implementierung (Systemrealisierung) fest.

Um diese Ziele zu erreichen, wird zuerst das System entworfen:

Das Gesamtsystem wird anhand der Anforderungsspezifikation in Teilsysteme zerlegt. Die relevanten Ausschnitte der Anforderungsspezifikation werden zu jedem Teilsystem zugeordnet und ein Software-Entwurf der zu realisierenden Teilsysteme findet statt.

Dieser Entwurf ist die Darstellung der Funktionen und Datenstrukturen in einer geeigneten Form, um danach diese Darstellung in einer Programmiersprache abzubilden. Erst jetzt wird die Programmiersprache festgelegt.

Die Vorbereitungen für den Beginn der nächsten Phase (Implementierungsphase) beinhalten die Festlegung der Realisierungsreihenfolge und natürlich die Termin- & Kostenplanung sowie die Benennung der Mitarbeiter, die die einzelnen Aufgaben übernehmen.

Phase 4: Systemrealisierung/Implementierung

In dieser Phase werden die Ergebnisse der Konzeption- und der Designphase umgesetzt. Oder einfacher gesagt: jetzt beginnt das Programmieren. Am Ende dieser Phase werden codierte und getestete Module vorliegen (jeder Programmierfachkraft wird eine Komponente zugeordnet).
Die dazugehörige Dokumentation (Moduldokumentation, Programmdokumentation und Testdokumentation) dient u. a. auch als Nachweis, dass alle Anforderungen, die in den vorherigen Phasen definiert wurden, erfüllt sind. Die Realisierung des Systems findet getrennt vom produktiven System, in einer Rechnerlandschaft, die das produktive System simuliert (Entwicklungsumgebung).

Phase 5: Systemtest/Systemeinführung:
Während der gesamten Realisierungsphase (oder sogar noch früher) wird getestet. Die Module werden getestet, Komponenten- & Integrationstests werden durchgeführt. Dadurch wird die Funktionalität und die Lauffähigkeit der Module innerhalb einer Komponente überprüft und die Zusammenarbeit der Komponenten getestet.
Jetzt wo aber das System fertig sein soll, wird es gegen die gesamten Anforderungen (die funktionalen und nicht funktionalen Anforderungen) getestet. Der Systemtest dient der Abnahme der Software durch den Kunden.
Erst nach der Überführung in den produktiven Betrieb kann die Schlussabnahme stattfinden. Die Überführung in den produktiven Betrieb kann entweder auf einen Schlag oder schrittweise stattfinden.

Phase 6: Systemtest Nutzung & Wartung:
Nach der erfolgreichen Softwareeinführung wird das System bewertet und optimiert. Obwohl es vor der Einführung getestet wurde, können neue Fehler auftauchen. Neue Releases werden eingespielt. Wichtig ist dabei die Software-Dokumentation aktuell zu halten.

Diese besteht aus:
- Programmierdokumentation (Quellcode)
- Methodendokumentation (allgemeine Beschreibung der Grundlagen, auf denen die Software beruht)
- Installationsdokumentation (für die Administratoren)
- Benutzerdokumentation (für die User)
- Datendokumentation (Datenstruktur, Schnittstellen usw.)
- Testdokumentation (Testbeschreibung)
- Entwicklungsdokumentation (Versionsmanagement)

Fachbegriff Prototyp

Ein Prototyp ist ein funktionsfähiges, aber vereinfachtes Versuchsmodell eines geplanten Produktes, eines Bauteils oder einer Software. Der Prozess zur Erstellung von Prototypen wird als Prototypenentwicklung bzw. Prototyping bezeichnet. Das Ziel beim Protoyping liegt in der Gewinnung von frühzeitigem Feedback bezüglich der Eignung eines Lösungsansatzes. Dass ein Prototyp meinst nur äußerlich – in Form und Größe – oder inhaltlich – im Sinne der genutzten Technik – einem möglichen Endprodukt entspricht, ist meist beabsichtigt.

In der Softwareentwicklung lassen sich durch einen Prototyp häufig Änderungswünsche frühzeitig erkennen und Probleme kostengünstiger beheben als bei einer vollumfänglichen Entwicklung.

Prototyp Arten in der Softwareentwicklung:
Grundsätzlich lassen sich bei der Entwicklung von Software unterschiedliche Arten von Prototypen bzw. Prototypings beschreiben:

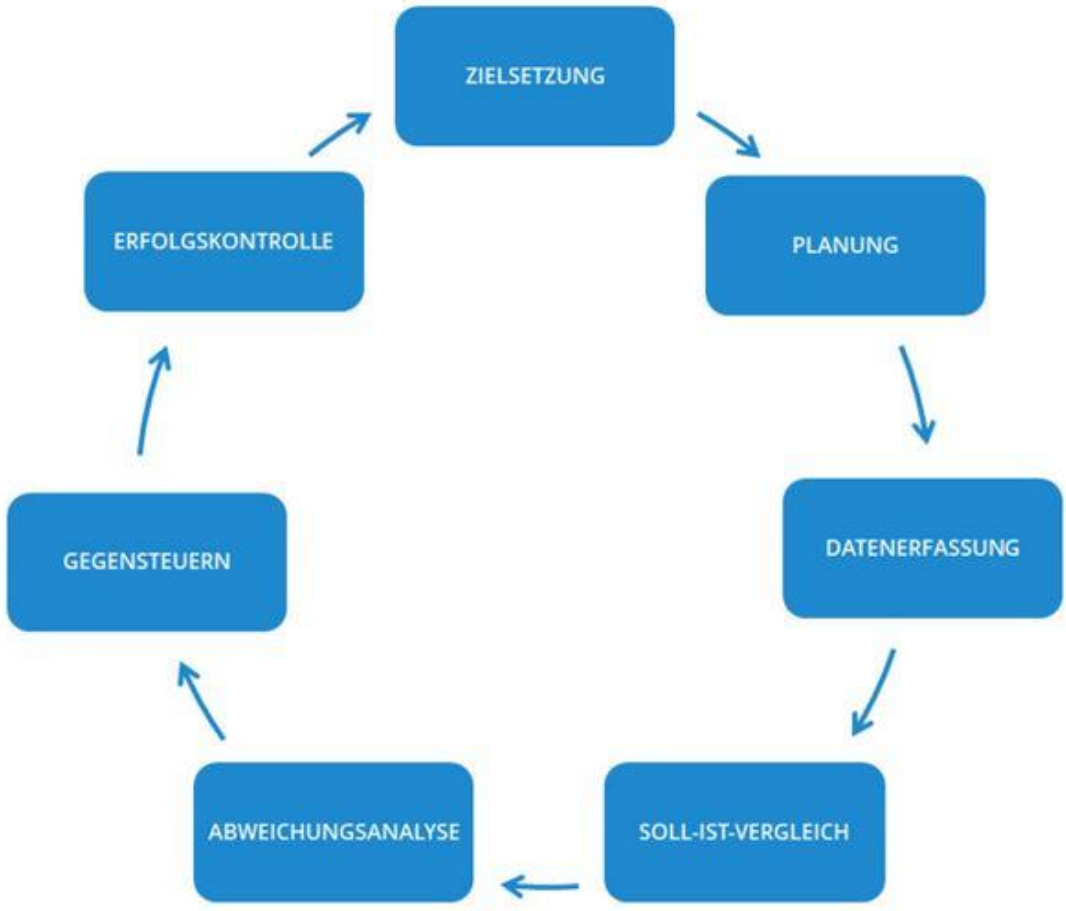
- Beim explorativen Protoyping gilt es nachzuweisen, ob eine Idee oder Spezifikation belastbar ist. Damit lassen sich Anforderungen ermitteln und Lösungsoptionen identifizieren, die später in einer Spezifikation oder in einem Lastenheft festgehalten werden können.
- Der evolutionäre Prototyp adressiert die Anwenderakzeptanz und verfolgt das Ziel, fehlende Funktionen zu identifizieren. Ein Klickdummy ist bspw. ein solch evolutionärer Prototyp.

- Der experimentelle Prototyp wird zu Forschungszwecken auf der Suche nach Möglichkeiten zur Realisierung genutzt.

Fachbegriff Soll-Ist-Analyse

Abgleich des geplanten und des tatsächlichen Zustandes

Der Soll-Ist-Vergleich wird zwischen Istwerten und -leistungen und den erwarteten Werten durchgeführt und ist eine Form des Projektcontrollings. Der Prozess, durch den der Gesamtunterschied zwischen den geplanten und den tatsächlichen Ergebnissen analysiert wird, ist auch als Abweichungsanalyse bekannt. Sind die Ereignisse besser als geplant, sind die Abweichungen positiv. Im anderen Fall sind sie negativ, d. h. Zielunterschreitung. Die Erkenntnisse dieser Analyse werden beim zukünftigen Planungsprozess berücksichtigt, um weitere Planabweichungen zu vermeiden.



Der Soll-Ist-Vergleich ist ein wichtiger Bestandteil des Projektcontrollings.

Die Abweichungsanalyse wird in Teilabweichungen zerlegt. Dabei sind die folgenden Ablaufschritte zu beachten. Zuerst werden die erreichten Ist-Werte den Soll-Werten gegenübergestellt und mit Hilfe der Abweichungsanalyse werden die Korrekturentscheidungen und Gegenmaßnahmen vorgeschlagen.

Während des Korrekturprozesses lässt diese Analyse entweder ursprüngliche Unternehmensziele realisieren oder führt zu neuen Unternehmenszielen. Dabei werden die Abweichungen erklärt und somit eine Grundlage für die Korrekturmaßnahmen geschaffen.

Fachbegriff Versionsverwaltung

Die Versionsverwaltung (Abgekürzt: VCS - Version Control System) ist ein System, das die Änderungen protokolliert, die an einer Datei oder einer Reihe von Dateien über die Zeit hinweg getätigt wurden. So ist es auch später immer möglich, auf eine vorherige Version zurückzugreifen, da jede archivierte Version mit einem Zeitstempel und einer Benutzerkennung gespeichert wird.

Zum Einsatz kommt die Versionsverwaltung insbesondere in der Softwareentwicklung zur Verwaltung der Quelltexte. Doch auch bei Büroanwendungen und bei Content-Management-Systemen wird kaum mehr auf die Versionsverwaltung verzichtet.

Wie funktioniert eine Versionsverwaltung:

Das System der Versionsverwaltung ist in der Lage, Arbeitskopien (organisiert in einem Verzeichnisbaum) mit den Daten aus dem Repository (engl. Aufbewahrungsort) zu synchronisieren. Das ist nötig, da es nur so möglich ist, mit den im Repository abgelegten Dateien zu arbeiten. Wird eine Version aus dem Repository in die Arbeitskopie übertragen, wird von einem Check-out, von Aus-Checken oder von Aktualisieren gesprochen. Die umgekehrte Übertragung von der Arbeitskopie in das Repository hingegen wird als Check-in, als Einchecken oder als Commit bezeichnet. Nun können die im Repository abgelegten Dateien mit diversen Programmen bearbeitet werden. Hierbei kann es sich um kommandozeilenorientierte Programme oder um solche mit grafischer Benutzeroberfläche handeln.

Was sind die Vorteile einer Versionsverwaltung:

Die Versionsverwaltung macht es möglich, Änderungen an einer Datei oder an einer Gruppe von Dateien zu speichern. Allerdings wird hier nicht etwa nur die Version nach der letzten Änderung gespeichert. Zunächst wäre da die Ausgangsversion einer Datei, die gesichert wird. Wird nun eine Änderung vollzogen, wird auch diese Änderung gespeichert, allerdings bleibt gleichzeitig auch die Ausgangsversion bestehen. So ist es für den Nutzer möglich, immer auf eine vorherige Version zurückzugreifen, wenn es Probleme mit der aktuellen Version gibt. Gleichzeitig ist die Versionsverwaltung aber auch essenziell, um überhaupt mit den Dateien arbeiten zu können. Das System sorgt nämlich dafür, dass der aktuelle (sowie der ältere) Stand eines Projekts, das in Form eines Verzeichnisbaums gespeichert ist, mit dem Repository synchronisiert wird.

Derzeit gibt es insgesamt drei Arten der Versionsverwaltung.
Das System ist hier entweder:

- lokal – Lokale Versionsverwaltungssysteme wurden mit Werkzeugen wie RCS und SCCS umgesetzt. Sie funktionieren nur auf einem Computer. Hinzu kommt, dass die lokale Versionsverwaltung meist nur eine einzige Datei versioniert. Anwendung findet die lokale Versionsverwaltung vor allem in Büroanwendungen. Hier werden die Versionen eines Dokuments dann direkt in der Datei des Dokuments gespeichert. In technischen Zeichnungen hingegen erfolgt die Versionsverwaltung hingegen etwa durch einen Änderungsindex.
- zentral – Die zentrale Versionsverwaltung ist als Client-Server-System aufgebaut und erlaubt so den netwerkweiten Zugriff auf ein Repositry. Mithilfe einer Rechteverwaltung kann dafür gesorgt werden, dass nur berechtigte Personen eine neue Version in das Archiv legen können. Populär wurde dieses Konzept durch das Open-Source-Projekt Concurrent Versions System (CVS). Neu implementiert hingegen wurde es mit Subversion (SVN) und wird inzwischen von vielen kommerziellen Anbietern eingesetzt.

- verteilt – Die verteilte Versionsverwaltung hingegen verwendet kein zentrales Repository. So verfügt jeder, der an einem Projekt arbeitet, ein eigenes Repository, dass er mit jedem x-beliebigen anderem Repository abgleichen kann. So ist die Versionsgeschichte genauso verteilt wie in den zentralen Verwaltungssystemen. Der Vorteil ist allerdings, dass Änderungen auch lokal erfolgen können – ganz ohne den Aufbau einer Server-Verbindung.

Ihre Gemeinsamkeit:

alle drei Generationen speichert üblicherweise nur die Unterschiede zwischen zwei Versionen. So lässt sich Speicherplatz sparen.