

Fragen zu Software Engineering allgemein:

Was ist UML? Nennen Sie die wichtigsten UML-Diagramme!

UML = **U**nified **M**odeling **L**anguage

Die Unified Modeling Language (UML) ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme.

Es handelt sich um eine Vereinheitlichung der grafischen Darstellung und Semantik der Modellierungselemente – es wird jedoch keine Methodik beschrieben.

Mit ihrer Hilfe lassen sich Analyse und Design im Softwareentwicklungsprozess beschreiben.

Grundsätzlich werden zwei Hauptdiagrammtypen unterschieden: Strukturdiagramme und Verhaltensdiagramme.

Strukturdiagramme

- Klassendiagramm (Klassen, Beziehungen untereinander, Merkmale)
- Komponentendiagramm (Komponenten, Verdrahtung)
- Verteilungsdiagramm (Verteilung von Artefakten auf Knoten)
- Profildiagramm (Stereotypen)
- Objektdiagramm (Exemplare von Klassen mit beispielhaften Werten)
- Kompositionsstrukturdiagramm (Innerer Aufbau von Klassen oder Objekten)
- Paketdiagramm (Ablagestruktur, Abhängigkeiten)
- Anwendungsfalldiagramm (Anwendungsfälle, Beziehungen zu Akteuren)

Verhaltensdiagramme

- Aktivitätsdiagramm (Aktionen, Flüsse, Verzweigungen)
- Interaktionsdiagramm (zulässige Reihenfolge von Aktivitäten)
 - o Interaktionsübersicht (Übersicht über mehrere Interaktionen)
 - o Kommunikationsdiagramm (Topologie des Nachrichtenaustauschs)
 - o Sequenzdiagramm (Reihenfolge des Nachrichtenaustauschs)
 - o Zeitdiagramm (zeitlicher Ablauf des Nachrichtenaustauschs)
- Zustandsdiagramm (Zustände, Übergänge, Verhalten)
 - o Protokollautomat (Zulässige Reihenfolge von Operationen)

Anwendungsfalldiagramm

Ein Anwendungsfalldiagramm entsteht meist während der Anforderungsphase und beschreibt die Geschäftsprozesse, indem es die Interaktion von Personen – oder von bereits existierenden Programmen – mit dem System darstellt. Die handelnden Personen oder aktiven Systeme werden Akteure genannt und sind im Diagramm als kleine Männchen angedeutet. Anwendungsfälle beschreiben dann eine Interaktion mit dem System.

Klassendiagramm

Für die statische Ansicht eines Programmentwurfs ist das Klassendiagramm einer der wichtigsten Diagrammtypen. Ein Klassendiagramm stellt zum einen die Elemente der Klasse dar, also die Attribute und

Operationen, und zum anderen die Beziehungen der Klassen untereinander. Klassen werden als Rechteck dargestellt, die Beziehungen zwischen den Klassen werden durch Linien angedeutet.

Objektdiagramm

Ein Klassendiagramm und ein Objektdiagramm sind sich auf den ersten Blick sehr ähnlich. Der wesentliche Unterschied besteht aber darin, dass ein Objektdiagramm die Belegung der Attribute, als den Objektzustand, visualisiert. Dazu werden sogenannte Ausprägungsspezifikationen verwendet. Mit eingeschlossen sind die Beziehungen, die das Objekt zur Laufzeit mit anderen Objekten hält.

Beschreibt zum Beispiel ein Klassendiagramm eine Person, so ist es nur ein Rechteck im Diagramm. Hat diese Person jedoch Assoziationen zu anderen Personen-Objekten, so können sehr viele Personen in einem Objektdiagramm verbunden sein, während ein Klassendiagramm diese Ausprägung nicht darstellen kann.

Sequenzdiagramm

Das Sequenzdiagramm stellt das dynamische Verhalten von Objekten dar. So zeigt es an, in welcher Reihenfolge Operationen aufgerufen und wann neue Objekte erzeugt werden. Die einzelnen Objekte bekommen eine vertikale Lebenslinie, und horizontale Linien zwischen den Lebenslinien der Objekte beschreiben die Operationen oder Objekterzeugungen. Das Diagramm liest sich somit von oben nach unten.

Beschreiben Sie die wichtigsten Tätigkeiten, die im Zuge eines Software-Projekts anfallen!

Was verstehen Sie unter Anforderungsanalyse?

Welche UML-Diagramme können in der Analysephase verwendet werden?

Was ist ein Sequenzdiagramm und wofür können Sequenzdiagramm eingesetzt werden?

Das Sequenzdiagramm ist ein Vertreter der Verhaltensdiagramme und gehört zur Untergruppe der Interaktionsdiagramme. Es wird nicht nur der momentane Zustand der Objekte abgebildet, sondern ein ganzer Ablauf. Sie zeigen den Ablauf von Objektinteraktionen.

Das Sequenzdiagramm stellt das dynamische Verhalten von Objekten dar. So zeigt es an, in welcher Reihenfolge Operationen aufgerufen und wann neue Objekte erzeugt werden. Die einzelnen Objekte bekommen eine vertikale Lebenslinie, und horizontale Linien zwischen den Lebenslinien der Objekte beschreiben die Operationen oder Objekterzeugungen. Das Diagramm liest sich somit von oben nach unten.

Im Besonderen steht der zeitliche Verlauf der Kommunikation im Vordergrund. Die Zeit läuft von oben nach unten. Sequenzdiagramme dürfen als Beteiligte Objekte und weitere Akteure beinhalten. Die Beteiligten kommunizieren über Nachrichten. Sie werden durch ein Rechteck mit Beschriftung, an dem unten eine vertikale, gestrichelte Linie angebracht ist, dargestellt. Die Gesamtheit aus Rechteck und gestrichelter Linie wird als Lebenslinie bezeichnet.

Rechtecke auf den gestrichelten Linien signalisieren, wann eine Lebenslinie aktiv ist. Methodenaufrufe und weitere Kommunikationsmöglichkeiten der Akteure untereinander werden durch Nachrichten dargestellt, die aus einer Beschreibung und einem waagerechten Pfeil bestehen

Eine synchrone Nachricht besitzt eine schwarze, gefüllte Pfeilspitze und wird mit einer Antwort (ungefüllte Pfeilspitze, gestrichelte Linie) quittiert. Synchron bedeutet in diesem Zusammenhang, dass die aufrufende Lebenslinie so lange blockiert, bis sie eine Antwort bekommt. Die Angabe des Antwortpfeils ist optional. Sie ist aber obligatorisch, wenn die Methode einen Wert zurück liefert.

Eine asynchrone Nachricht wird durch eine ungefüllte Pfeilspitze an einer durchgehenden Linie dargestellt. Asynchron bedeutet, dass der Sender der Nachricht nicht auf eine Antwort wartet und nicht blockiert.


Am besten verwendet man Sequenzdiagramme zur beispielhaften Veranschaulichung von Vorgängen. Sie bilden nichts Statisches ab, sondern etwas Dynamisches. Sequenzdiagramme werden immer dann gebraucht, wenn man Abläufe innerhalb von Softwaresystemen darstellen will. Mit ihnen ist man in der Lage die Kommunikation zwischen Klassen oder Objekten darzustellen.

Welche Tätigkeiten fasst man unter dem Begriff „Design“ zusammen?
--

Welche Beziehungen zwischen Klassen bzw. Objekten können in einem Klassendiagramm eingetragen werden?
--

Spezialisierung/Generalisierung


Eine Generalisierung/Spezialisierung ist eine Beziehung zwischen einem allgemeineren und einem spezielleren Element (bzw. umgekehrt), wobei das speziellere weitere Eigenschaften hinzufügt und sich kompatibel zum allgemeinen verhält. Vererbung ist ein Umsetzungsmechanismus für die Relation zwischen Ober- und Unterklasse, wodurch Attribute und Operationen der Oberklasse auch den Unterklassen zugänglich gemacht werden.

Notation: Die Vererbungsbeziehung wird mit einem großen, nicht gefüllten Pfeil dargestellt, wobei der Pfeil von der Unterklasse zur Oberklasse zeigt. 

Abhängigkeit

Eine Abhängigkeit ist eine Beziehung von einem (oder mehreren) Quellelement(en) zu einem (oder mehreren) Zielelement(en).


Die Zielelemente sind für die Spezifikation oder Implementierung der Quellelemente erforderlich.

Notation: Dargestellt wird eine Abhängigkeit durch einen gestrichelten Pfeil, wobei der Pfeil vom abhängigen auf das unabhängige Element zeigt. 

Realisierung

Die Realisierungsbeziehung ist eine spezielle Abstraktionsbeziehung. Es ist eine Beziehung zwischen einer Implementierung und ihrem Spezifikationselement.


Das abhängige Element implementiert das unabhängige Element (z.B. eine Schnittstelle oder ein abstraktes Element).

Notation: eine gestrichelte Linie mit einer dreieckigen Pfeilspitze, wobei der Pfeil vom abhängigen auf das unabhängige Element zeigt. 

Assoziation


Eine Assoziation beschreibt als Relation zwischen Klassen die gemeinsame Semantik und Struktur einer Menge von Objektverbindungen.

Gewöhnlich ist eine Assoziation eine Beziehung zwischen zwei Klassen. Grundsätzlich kann aber jede beliebige Anzahl von Klassen an einer Assoziation beteiligt sein.

Notation: Assoziationen werden durch eine Linie zwischen den beteiligten Klassen dargestellt. An den jeweiligen Enden der Linie wird die Multiplizität notiert. 

Gerichtete Assoziation

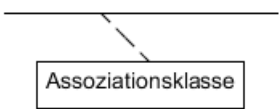
Eine gerichtete Assoziation ist eine Assoziation, bei der von der einen beteiligten Klasse zur anderen direkt navigiert werden kann, nicht aber umgekehrt.

Notation: Eine gerichtete Assoziation wird wie eine gewöhnliche Assoziation notiert, jedoch hat sie auf der Seite der Klasse, zu der navigiert werden kann eine offene Pfeilspitze. Die Richtung, in die nicht navigiert werden kann, wird durch ein kleines Kreuz auf der Seite der Klasse markiert, zu der nicht navigiert werden kann. 

Attributierte Assoziation


Eine attributierte Assoziation verfügt sowohl über die Eigenschaften einer Klasse als auch über die einer Assoziation. Es kann gesehen werden als eine Assoziation mit zusätzlichen Klasseneigenschaften oder als Klasse mit zusätzlichen Assoziationseigenschaften.

Wenn zwei Klassen in Beziehung zueinander stehen, kann es sein, dass es Eigenschaften gibt, die weder zu einer noch zur anderen Klasse gehören, sondern zur Beziehung zwischen den beiden. Mit einer Assoziationsklasse kann dies modelliert werden.

Notation: Attributierte Assoziationen werden wie gewöhnliche Assoziationen dargestellt, zusätzlich ist jedoch über eine gestrichelte Linie, die von der Assoziationslinie abgeht, ein normales Klassensymbol angehängt. 

Mehrgliedrige Assoziation

Eine mehrgliedrige Assoziation ist eine Assoziation, an der mehr als zwei Klassen beteiligt sind.

Notation: Eine mehrgliedrige Assoziation wird mit einer nicht ausgefüllten Raute gezeichnet, die größer ist als die Aggregationsraute. Die Klassen werden mit Linien mit der Raute verbunden. 

Qualifizierte Assoziation

Bei einer qualifizierten Assoziation wird die referenzierte Menge der Objekte durch qualifizierende Attribute in Partitionen unterteilt.

Die durch eine Assoziation spezifizierte Menge von verlinkten Objekten kann durch eine ihrer Eigenschaften in Untermengen (Partitionen) aufgeteilt werden. Diese Eigenschaft kann man als Qualifiziere modellieren.

Notation: Das für die Assoziation benutzte qualifizierende Attribut wird in einem Rechteck an der Seite der Klasse notiert, die über diesen Qualifizierer auf das Zielobjekt zugreift.



Aggregation

Eine Aggregation ist eine Assoziation, erweitert um den semantisch unverbindlichen Kommentar, dass die beteiligten Klassen keine gleichwertige Beziehung führen, sondern eine Ganzes-Teile-Hierarchie darstellen. Eine Aggregation soll beschreiben, wie sich etwas Ganzes aus seinen Teilen logisch zusammensetzt.

Kennzeichnend für alle Aggregationen ist, dass das Ganze Aufgaben stellvertretend für seine Teile wahrnimmt. Im Gegensatz zur Assoziation führen die beteiligten Klassen keine gleichberechtigten Beziehung, sondern eine Klasse (das Aggregat) bekommt eine besondere Rolle und übernimmt stellvertretend die Verantwortung und Führung.

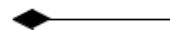
Notation: Eine Aggregation wird wie eine Assoziation als Linie zwischen zwei Klassen dargestellt und zusätzlich mit einer kleinen Raute versehen. Die Raute steht auf der Seite des Aggregats (des Ganzen).



Komposition

Eine Komposition ist eine strenge Form der Aggregation, bei der das Ganze verantwortlich ist für die Existenz und Speicherung der Teile. Sie beschreibt, wie sich etwas Ganzes aus Einzelteilen zusammensetzt und diese kapselt.

Notation: Die Komposition wird wie die Aggregation als Linie zwischen zwei Klassen gezeichnet und mit einer kleinen gefüllten Raute auf der Seite des Ganzen versehen.



Was steht in einem Projektplan? Was ist ein Meilenstein?

Projektplan

Der Projektplan, der zu Beginn des Projekts erstellt wird, soll das Projektteam und die Kunden über Arbeitsaufteilung und –ablauf informieren und helfen, den Fortschritt des Projekts einzuschätzen. Der Projektplan definiert die Aufteilung der Arbeit und die Ressourcen, die für das Projekt verfügbar sind. Der Plan sollte außerdem die Risiken ermitteln, die dem Projekt und der zu entwickelnden Software drohen, und aufzeigen, welcher Ansatz beim Risikomanagement verfolgt wird.

Der Projektplan umfasst normalerweise folgende Abschnitte:

Einleitung

Dies beschreibt kurz die Ziele des Projekts und definiert die Rahmenbedingungen (Finanzrahmen, Zeit), die das Projektmanagement beeinflussen.

Projektorganisation

Dieser Abschnitt beschreibt die Organisationsstruktur des Teams, die benötigten Mitarbeiter und ihre Rolle im Team.

Risikoanalyse

Diese Analyse beschreibt potenzielle Projektrisiken, ihre Eintrittswahrscheinlichkeit und mögliche Strategien zur Reduzierung der Risiken.

Anforderungen an Hardware- und Softwareressourcen

Dieser Abschnitt spezifiziert, welche Hard- und Software zur Entwicklung erforderlich sind. Falls Hardware angeschafft werden muss, können hier Schätzwerte für Preise und Liefertermine angegeben werden.

Arbeitsaufteilung

Hier wird das Projekt in Aktivitäten zerlegt und die Meilensteine und Lieferungen der einzelnen Aktivitäten beschrieben.

Projektzeitplan

Dieser Abschnitt zeigt die Abhängigkeiten zwischen den Aktivitäten, der geschätzten Zeit zum Erreichen der einzelnen Meilensteine und der Zuweisung von Mitarbeitern zu Aktivitäten.

Mechanismen zur Überwachung und Berichterstellung

Dieser Abschnitt definiert welche Berichte das Management zu erstellen hat, wann sie zu erstellen sind und welche Überwachungsmechanismen verwendet werden.

Meilensteine

Meilensteine sind wichtige Stadien im Projekt, in denen der Fortschritt bewertet werden kann.

Meilenstein ist ein „Ereignis besonderer Bedeutung“ im Ablauf eines Projekts. Wesentlicher Bestandteil eines Meilensteins ist oft die Termineinhaltung. Man kann einen Meilenstein als einen „nach außen kommunizierten Zeitpunkt, bis wann bestimmte Aufgaben erledigt sein müssen“ verstehen. Ein Meilenstein hat dabei keine zeitliche Ausprägung (Dauer = 0 Tage) und beinhaltet keine Tätigkeit⁴

Welche Methoden der Qualitätssicherung können in einem Software-Projekt eingesetzt werden?

Was versteht man unter Versionskontrolle?

Was beeinflusst den Aufwand eines Software-Projekts? Wie kann Aufwand geschätzt werden?

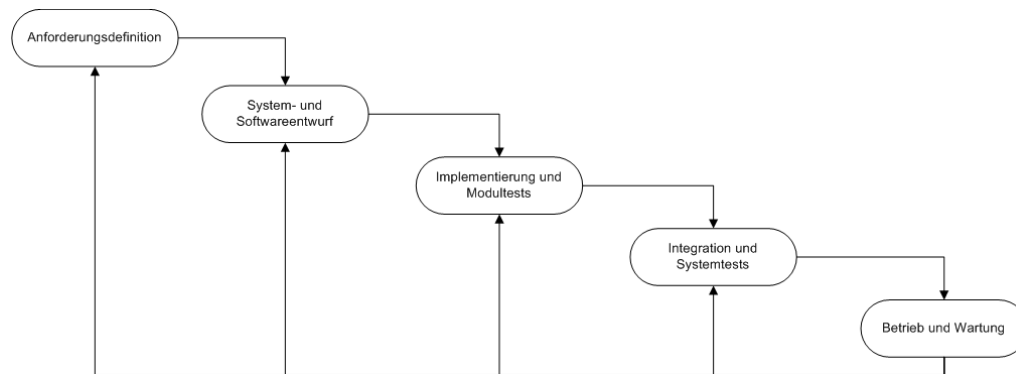
Welche Vorgehensmodelle der Software-Entwicklung gibt es?

Bei einem Vorgehensmodell handelt es sich um eine vereinfachte Darstellung eines Softwareprozesses.

Wasserfallmodell

Dieses Modell stellt die grundlegenden Prozessabläufe wie Spezifikation, Entwicklung, Validierung und Evolution als eigenständige Phasen dar, wie zum Beispiel Anforderungsspezifikation, Softwareentwurf, Implementierung, Test usw.

Wegen der Kaskade von einer Phase zur nächsten wird dieses Modell Wasserfallmodell oder Softwarelebenszyklus genannt.



Die wichtigsten Phasen des Wasserfallmodells spiegeln direkt die grundlegenden Entwicklungsaktivitäten wider:

- Analyse und Definition der Anforderungen
- System- und Softwareentwurf
- Implementierung und Modultest
- Integration und Systemtest
- Betrieb und Wartung

Im Prinzip gehen aus jeder Phase ein oder mehrere Dokumente hervor, die genehmigt oder abgenommen werden. Die nächste Phase sollte nicht beginnen, bevor nicht die vorherige abgeschlossen wurde. In der Praxis überlappen sich die Phasen und tauschen Informationen untereinander aus.

Das Wasserfallmodell ist konsistent zu anderen Systementwicklungsmodellen und erzeugt in jeder Phase Dokumentationen. Dadurch bleibt der Prozess durchschaubar, sodass Manager den Fortschritt gemäß Entwicklungsplan überwachen können.

Das Hauptproblem ist jedoch die starre Aufteilung des Projekts in verschiedene Phasen. Zu einem frühen Zeitpunkt müssen Verbindlichkeiten eingegangen werden, was es schwer macht, auf neue Anforderungen des Kunden zu reagieren.

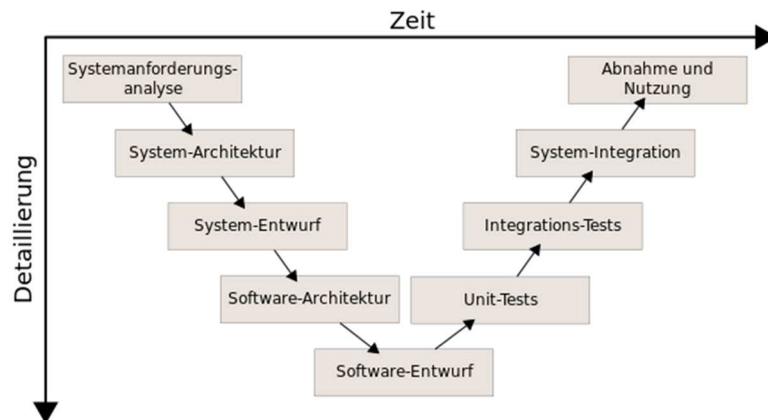
Im Prinzip sollt das Wasserfallmodell nur Verwendung finden, wenn die Anforderungen gut durchdacht sind und es eher unwahrscheinlich ist, dass es während der Systementwicklung zu gravierenden Änderungen kommt.

V-Modell

Das V-Modell ist ein Vorgehensmodell in der Softwareentwicklung, bei dem der Softwareentwicklungsprozess in Phasen organisiert wird. Neben diesen Entwicklungsphasen definiert das V-Modell auch das Vorgehen zur Qualitätssicherung (Testen) phasenweise.

Es basiert auf dem Wasserfallmodell: Die Phasenergebnisse sind bindende Vorgaben für die nächsttiefere Projektphase. Der linke, nach unten führende Ast für die Spezifizierungsphasen schließt mit der Realisierungsphase ab. Eine Erweiterung gegenüber dem Wasserfallmodell sind die zeitlich nachfolgenden Testphasen, die im rechten, nach oben führenden Ast dargestellt werden. Den spezifizierenden Phasen

stehen jeweils testende Phasen gegenüber, was in der Darstellung ein charakteristisches „V“ ergibt, das dem Modell auch den Namen gab. Diese Gegenüberstellung soll zu einer möglichst hohen Testabdeckung führen, weil die Spezifikationen der jeweiligen Entwicklungsstufen die Grundlage für die Tests (Testfälle) in den entsprechenden Teststufen sind.



Inkrementelle Entwicklung

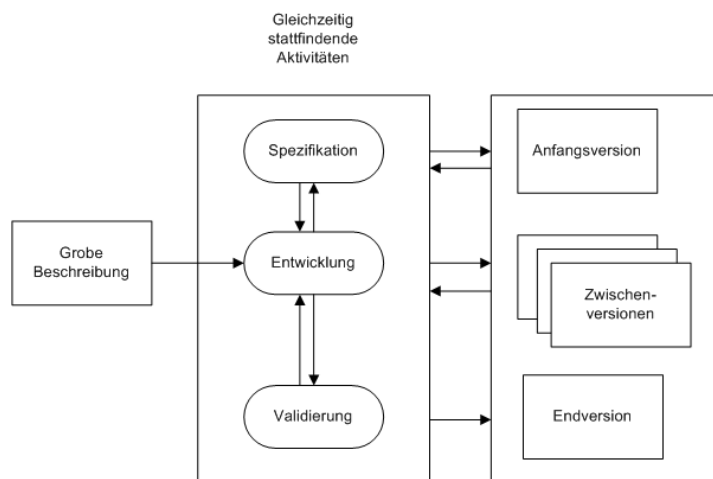
Dieser Ansatz verknüpft die Aktivitäten der Spezifikation, der Entwicklung und der Validierung. Das System wird als eine Folge von Versionen (Inkrementen) entwickelt, wobei jede Version neue Funktionalität zu der vorherigen hinzufügt.

Die inkrementelle Entwicklung basiert darauf, eine Anfangsimplementierung zu entwickeln, die Benutzer zu Kommentaren und Hinweisen zu dieser Implementierung aufzufordern und sie über mehrere Versionen hinweg zu verbessern, bis ein angemessenes System entstanden ist.

Die Spezifikation, die Entwicklung und die Validierung werden nicht als separate Abläufe betrachtet, sondern werden gleichzeitig ausgeführt, wobei sie untereinander Rückmeldungen zügig austauschen.

Inkrementelle Software-Entwicklung ist ein fundamentaler Teil des agilen Ansatzes; sie ist besser als ein Wasserfallansatz für die meisten Geschäftsfälle, E-Commerce und individuellen Systeme geeignet.

Jedes Inkrement oder jede Version des Systems besitzt einen Teil der Funktionalität, die vom Kunden gebraucht wird. In der Regel enthalten die frühen Systeminkremente die wichtigsten oder am dringendsten benötigten Funktionen. Dies bedeutet, dass der Kunde das System zu einem relativ frühen Zustand in der Entwicklung evaluieren kann um festzustellen, ob es den Anforderungen entspricht.



Inkrementelle Entwicklung hat gegenüber dem Wasserfallmodell drei wesentliche Vorteile

- Die Kosten für die Anpassung an sich ändernde Kundenanforderungen werden reduziert. Der Umfang der wiederholt durchzuführenden Analyse und Dokumentation ist geringer als beim Wasserfallmodell.
- Es ist einfacher, Rückmeldungen der Kunden zu bereits fertiggestellten Teilen der Entwicklungsarbeit zu bekommen. Sie können sich bei Softwaredemonstrationen äußern und sehen, wie viel implementiert wurde.
- Eine schnellere Auslieferung und Installation von verwendungsfähiger Software an den Kunden ist selbst dann möglich, wenn noch nicht die gesamte Funktionalität enthalten ist. Die Kunden können die Software früher verwenden und daraus Nutzen ziehen, als es mit einem Wasserfallmodell möglich wäre.

Inkrementelle Entwicklung ist heute die am häufigsten eingesetzte Vorgehensweise für die Entwicklung von Anwendungssystemen. Dieser Ansatz kann entweder plangesteuert, agil oder – am häufigsten jedoch – eine Mischung dieser Methoden sein.

In einem plangesteuerten Ansatz werden die Systeminkremente im Voraus bestimmt. Bei einem agilen Vorgehen werden zwar die frühen Inkremente ermittelt, die Entwicklung der späteren Inkremente hängt jedoch vom Projektfortschritt und von den Prioritäten des Kunden ab.

Die inkrementelle Entwicklung hat allerdings aus Managementsicht zwei Schwachstellen:

- Der Prozess ist nicht sichtbar. Manager brauchen in regelmäßigen Abständen Zwischenversionen, an denen sie den Fortschritt messen können. Wenn Systeme schnell entwickelt werden, ist es nicht kosteneffektiv, jede Version zu dokumentieren.
- Die Systemstruktur wird tendenziell schwächer, wenn neue Inkremente hinzugefügt werden. Solange nicht Zeit und Geld in die Überarbeitung zur Verbesserung der Software investiert wird, besteht die Tendenz, dass stetige Veränderungen die Struktur der Software beeinträchtigen. Die Integration von weiteren Softwareänderungen wird zunehmen schwerer und teurer. Bei großen, komplexen Systemen mit einer langen Lebensdauer werden die Probleme der inkrementellen Entwicklung schnell akut, wenn verschiedene Teams verschiedene Teile des Systems entwickeln. Große Systeme brauchen einen stabilen Rahmen oder Architektur und Verantwortlichkeiten von unterschiedlichen Teams, die an Teilen der Software arbeiten, müssen bezüglich dieser Architektur klar festgelegt werden. Dies muss im Voraus geplant statt inkrementell entwickelt werden. Man kann ein System inkrementell entwickeln und es den Kunden zum Kommentieren vorführen, ohne es wirklich auszuliefern und in der Umgebung des Kunden einzurichten. Inkrementelle Auslieferung und Einrichtung bedeutet, dass die Software in realen betrieblichen Prozessen verwendet wird. Dies ist nicht immer möglich, da das Ausprobieren der neuen Software die normalen Geschäftsfälle stören kann.

Obwohl die inkrementelle Entwicklung viele Vorteile hat, so ist sie doch nicht ganz ohne Probleme. Der Hauptgrund für Schwierigkeiten ist die Tatsache, dass große Organisationen bürokratische Handlungsabläufe besitzen, die sich im Laufe der Zeit entwickelt haben und die unter Umständen nicht mit einem informelleren iterativen oder agilen Prozess harmonisieren.

Manchmal gibt es auch einen guten Grund für diese Handlungsabläufe – zum Beispiel um sicherzustellen, dass die Software gesetzliche Regelungen ordnungsgemäß umsetzt. Diese Geschäftsabläufe zu verändern ist nicht immer möglich, sodass Konflikte zwischen den Prozessen unvermeidbar sind.

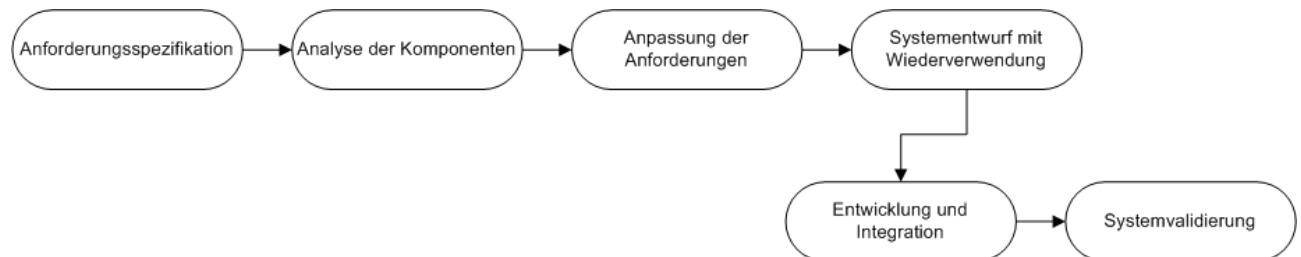
Wiederverwendungsorientiertes Software-Engineering

Dieses Modell basiert auf der Existenz einer beträchtlichen Anzahl von wiederverwendbaren Komponenten. Der Systementwicklungsprozess beschäftigt sich mehr damit, diese Komponenten in ein System zu integrieren, als damit, neue Komponenten von Grund auf zu entwickeln.

In einem Großteil aller Softwareprojekte wird Software wiederverwendet. Häufig geschieht das informell, wenn die Mitarbeiter eines Projekts von einem Entwurf oder von Code wissen, der dem ähnelt, der

gebraucht wird. Sie suchen ihn heraus, verändern ihn nach ihren Bedürfnissen und bauen ihn in ihr System ein. Diese informelle Wiederverwendung findet unabhängig von dem eingesetzten Entwicklungsprozess statt.

Im 21. Jahrhundert wurden jedoch Vorgehensmodelle der Softwareentwicklung immer populärer, die den Schwerpunkt auf die Wiederverwendung bereits vorhandener Software legten. Wiederverwendungsorientierte Ansätze beruhen auf einer großen Menge wiederverwendbarer Softwarekomponenten und auf einem Integrationsrahmen für die Zusammenstellung dieser Komponenten. Manchmal handelt es sich bei diesen Komponenten um eigenständige käufliche Systeme (COTS-Systeme, Commercial Off-The-Shelf System), die benutzt werden können, um eine spezielle Funktion beizusteuern.



Obwohl die erste Phase der Anforderungsspezifikation und die Validierungsphase auch in anderen Softwareprozessen vorkommen, sind die Zwischenstufen in einem wiederverwendungsorientierten Prozess andere. Diese Stufen sind:

- Analyse der Komponenten
- Anpassung der Anforderungen
- Systementwurf und Wiederverwendung
- Entwicklung und Integration

Es gibt drei Arten von Softwarekomponenten, die in einem wiederverwendungsorientierten Prozess eingesetzt werden können

- Webdienste, die im Hinblick auf Servicestandards entwickelt werden und für entfernte Aufrufe verfügbar sind
- Sammlungen von Objekten, die als Pakete entwickelt werden, um mit Komponenten-Frameworks wie .NET oder J2EE integriert zu werden
- Eigenständige Softwaresysteme, die für die Benutzung in einer bestimmten Umgebung konfiguriert wurden.

Wiederverwendungsorientiertes Software-Engineering hat den offensichtlichen Vorteil, dass es die Menge an zu entwickelnder Software und somit auch die Kosten und Risiken minimiert. Außerdem wird die Software schneller geliefert.

Kompromisse bei den Anforderungen sind jedoch unvermeidbar, und das kann zu einem System führen, das die wirklichen Bedürfnisse des Benutzers nicht erfüllt.

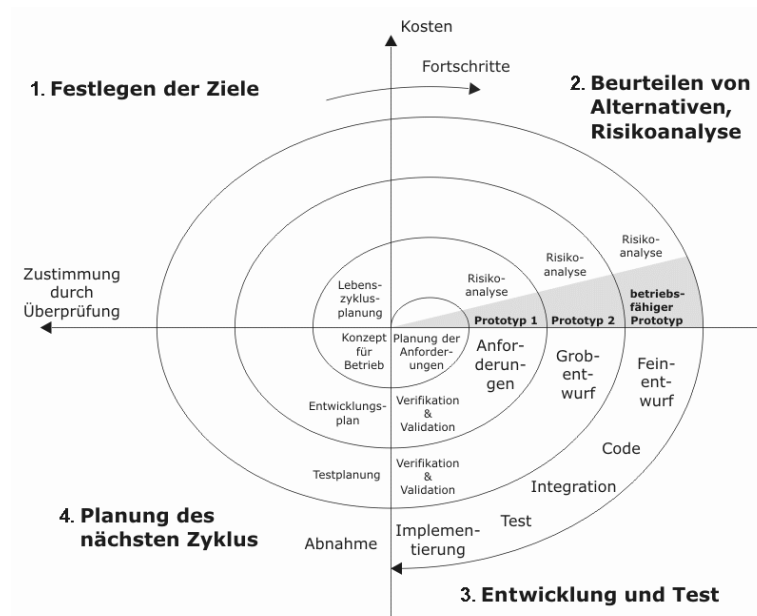
Außerdem geht ein Teil der Kontrolle über die Weiterentwicklung des Systems verloren, da sich neue Versionen wiederverwendeter Komponenten der Kontrolle der Organisation entziehen, die sie benutzt.

Spiralmodell nach Boehm

Der Softwareprozess wird als Spirale dargestellt anstatt als eine Folge von Aktivitäten mit Rückwärtsbezügen von einer Aktivität zur anderen. Jede Windung der Spirale steht für eine Phase des

Prozesses. So beschäftigt sich die innere Windung mit der Machbarkeit des Systems, die nächste mit der Definition der Systemanforderungen, die folgende mit dem Systementwurf, usw.

Das Spiralmodell kombiniert die Vermeidung von Änderungen mit Änderungstoleranz. Es geht davon aus, dass Änderungen ein Ergebnis von Projektrisiken sind und beinhaltet explizite Risikomanagementaktivitäten, um diese Risiken zu reduzieren.



Jede Spirale ist in vier Segmente aufgeteilt:

- Ziele aufstellen
- Risiken einschätzen und verringern
- Entwicklung und Validierung
- Planung

Der Hauptunterschied zwischen dem Spiralmodell und anderen Vorgehensmodellen liegt in der ausdrücklichen Betrachtung der Risiken im Spiralmodell.

Rational Unified Process

Der Rational Unified Process (RUP) vereinigt Elemente aus allen allgemeinen Vorgehensmodellen, veranschaulicht empfohlene Vorgehensweisen für Spezifikation und Entwurf und unterstützt die Entwicklung von Prototypen und die inkrementelle Auslieferung.

Der RUP trägt der Tatsache Rechnung, dass herkömmliche Vorgehensmodelle nur eine einzige Sicht des Prozesses darstellen. Der RUP wird hingegen normalerweise aus drei Perspektiven heraus beschrieben:

- Eine dynamische Perspektive, die die Phasen des Modells zeitlich darstellt
- Eine statische Perspektive, die die ausgeführten Prozessaktivitäten darstellt
- Eine praxisbezogene Perspektive, die die während des Prozesses empfohlene Vorgehensweise vorschlägt

Beim RUP handelt es sich um ein phasenorientiertes Modell, das für den Softwareprozess vier einzelne Phasen unterscheidet. Anders als beim Wasserfallmodell, bei dem die Phasen mit Prozessaktivitäten gleichgesetzt werden, sind die Phasen des RUP enger mit geschäftlichen als mit fachlichen Belangen verbunden.

Die Phasen lauten wie folgt:

- Konzeption
- Entwurf
- Konstruktion
- Übergabe

Die Wiederholung innerhalb des RUP wird auf zwei Arten unterstützt. Jede Phase kann iterativ ausgeführt werden, wobei die Ergebnisse schrittweise entwickelt werden. Die Phasen können ebenfalls insgesamt iterativ ausgeführt werden.

Die statische Sicht des RUP konzentriert sich auf die Aktivitäten während des Entwicklungsprozesses. Diese werden in der RUP-Beschreibung als Arbeitsabläufe bezeichnet.

Der Prozess legt sechs Hauptarbeitsabläufe und drei unterstützende Arbeitsabläufe fest.

- Geschäftsprozessmodellierung
- Anforderungsanalyse
- Analyse und Entwurf
- Implementierung
- Tests
- Auslieferung
- Konfigurations- und Änderungsmanagement.
- Projektmanagement
- Infrastruktur

Der Vorteil von dynamischen und statischen Sichten liegt darin, dass die Phasen des Entwicklungsprozesses nicht mit speziellen Arbeitsabläufen verbunden sind. Zumindest in der Theorie können alle RUP-Arbeitsabläufe in allen Phasen des Prozesses aktiv sein.

Die praxisbezogene Perspektive des RUP beschreibt empfohlene Vorgehensweisen für das Software-Engineering zum Einsatz bei der Systementwicklung.

Sechs grundlegende Vorgehensweisen werden empfohlen:

- Software iterativ zu entwickeln
- Anforderungen verwalten (eindeutige Dokumentation, Verfolgung von geänderten Anforderungen)
- Komponentenbasierende Architekturen verwenden (Aufteilung der Systemarchitektur in Komponenten)
- Software visuell modellieren (Verwenden grafischer UML-Modelle)
- Softwarequalität verifizieren
- Änderungen der Software steuern

AGILE METHODEN

Agile Methoden sind inkrementelle Entwicklungsmethoden, bei denen die Inkremente klein sind und bei denen typischerweise alle zwei bis drei Wochen neue Versionen des Systems erzeugt werden und dem Kunden zur Verfügung gestellt werden. Sie beziehen Kunden in den Entwicklungsprozess ein, um schnellere Rückmeldungen auf sich ändernde Anforderung zu bekommen. Indem eher informelle Kommunikation eingesetzt wird anstatt formelle Sitzungen mit schriftlichen Unterlagen abzuhalten, wird die Dokumentation minimiert.

Prinzipien der agilen Methoden:

- Einbeziehung des Kunden
- Inkrementelle Auslieferung

- Menschen statt Prozesse
- Offen für Änderungen
- Einfachheit

Extreme Programming (XP)

Extreme Programming ist wahrscheinlich die bekannteste und am häufigsten verwendete agile Methode. Dieser Ansatz wurde dadurch entwickelt, bekannte gute Praktiken wie iterative Entwicklung „ins Extreme“ zu steigern. Beim Extreme Programming werden Anforderungen in Form von Szenarios ausgedrückt, die User-Stories genannt werden. Diese Szenarios werden direkt in Form einer Abfolge von Aufgaben implementiert. Die Programmierer arbeiten paarweise zusammen, um Tests für die einzelnen Aufgaben zu entwickeln, bevor sie Code schreiben. Alle Tests müssen erfolgreich ausgeführt werden, wenn neuer Code in das System integriert wird. Zwischen den einzelnen Releases des Systems vergeht nur wenig Zeit.

Extreme Programming umfasst eine Reihe von Vorgehensweisen, welche die Prinzipien agiler Methoden widerspiegeln

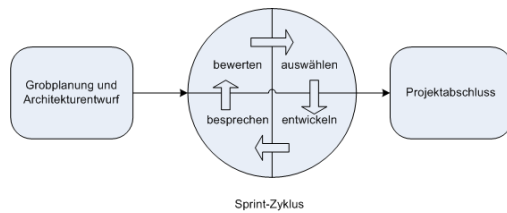
- Die inkrementelle Entwicklung wird durch kleine häufige Releases des Systems erreicht
- Die Einbeziehung des Kunden wird durch die kontinuierliche Teilnahme eines Kundenbevollmächtigten an der Entwicklung erreicht.
- Durch Paarprogrammierung, kollektives Eigentum am Systemcode und einen geeigneten Entwicklungsprozesses, bei dem es keine extrem langen Arbeitszeiten gibt, werden Menschen statt Prozesse in den Vordergrund gerückt.
- Änderungen werden durch regelmäßige Systemreleases an die Kunden, Test-First-Entwicklung, Refactoring zur Vermeidung von Codedegeneration und stetige Integration von neuen Funktionalitäten eingebunden.
- Der Erhalt der Einfachheit wird durch ständiges Refactoring zur Verbesserung der Codequalität und die Verwendung einfacher Entwürfe erreicht, die zukünftigen Änderungen am System nicht unnötigen vorgreifen.

Vorgehensweise

- Inkrementelle Planung (Anforderungen werden auf Story-Cards aufgezeichnet. Welche User-Stories in ein Release aufgenommen werden, wird aufgrund der verfügbaren Zeit und der jeweiligen Priorität entschieden. Die Entwickler teilen die User-Stories in Entwicklungsaufgaben auf)
- Kleine Releases
- Einfacher Entwurf
- Test-First-Entwicklung
- Refactoring (von allen Entwicklern wird erwartet, dass sie den Code einem kontinuierlichen Refactoring unterziehen, sobald Verbesserungsmöglichkeiten entdeckt werden)
- Paarprogrammierung
- Kollektives Eigentum (Entwickler arbeiten an allen Bereichen des Systems, keine Experten-Inseln)
- Kontinuierliche Integration
- Erträgliche Arbeitsgeschwindigkeit (große Mengen an Überstunden werden als nicht akzeptabel betrachtet)
- Kunde vor Ort (ein Bevollmächtigter des Kunden soll dem XP-Team ständig zur Verfügung stehen)

Scrum

Der Scrum-Ansatz ist eine Methode, die sich eher auf die Verwaltung iterativer Entwicklung als auf spezifische technische Aspekte des agilen Software-Engineerings konzentriert. Scrum schreibt nicht die Benutzung von bestimmten Programmiermodellen und testgetriebener Entwicklung vor.



Es gibt drei Phasen in Scrum:

- Allgemeine Planungsphase
- Serie von Sprint-Zyklen
- Projektabschluss

Das innovative an Scrum ist seine zentrale Phase, die Sprint-Zyklen. Ein Sprint in Scrum ist eine Planungseinheit, in der die auszuführende Arbeit abgeschätzt wird, die zu entwickelnden Leistungsmerkmale ausgesucht werden und die Software implementiert wird. Am Ende eines Sprints wird die vollständige Funktionalität an die Projektbeteiligten ausgeliefert.

Die Hauptcharakteristika dieses Prozesses sind:

- Sprints haben eine feste Länge (normalerweise 2 – 4 Wochen)
- Ausgangsbasis für die Planung ist das Produkt-Backlog: eine Liste der Aufgaben, die erfüllt werden muss. Während der Bewertungsphase des Sprints wird diese Liste überprüft
- In der Auswahlphase sind alle Mitglieder des Projektteams einbezogen, um die Merkmale und Funktionalitäten auszuwählen, die entwickelt werden sollen
- Nachdem die Auswahl getroffen wurde, organisiert sich das Team zur Entwicklung der Software selbst. Es werden täglich kurze Treffen mit allen Teammitgliedern abgehalten. Während dieser Phase ist das Team vom Kunden und dem Unternehmen isoliert, die gesamte Kommunikation wird über den sogenannten Scrum-Master gelenkt. Die Rolle des Scrum-Masters ist es, das Entwicklerteam von äußeren Ablenkungen fernzuhalten.
- Am Ende des Sprints wird die Arbeit einer Besprechung unterzogen und den Projektbeteiligten vorgestellt. Der nächste Sprint-Zyklus beginnt.

Das Konzept von Scrum ist, dass das gesamte Team befugt ist, Entscheidungen zu fällen. Daher wurde der Begriff Projektmanager bewusst vermieden. Stattdessen fungiert der Scrum-Master als eine Art Moderator.

Folgende Vorteile von Scrum werden aufgeführt:

- Das Produkt wird in eine Reihe von verwaltbaren und verständlichen Teilstücken zerlegt
- Instabile Anforderungen behindern den Fortschritt nicht
- Jedes Projektmitglied hat Sicht auf das gesamte System, Teamkommunikation wird dadurch gefördert
- Kunden bekommen die einzelnen Inkremente fristgerecht geliefert und erhalten Feedback über die Funktionsweise des Produkts
- Zwischen Kunden und Entwicklern wird Vertrauen aufgebaut und eine positive Atmosphäre wird erzeugt, in der jeder davon ausgeht, dass das Projekt gelingt

Scrum wurde ursprünglich für Teams entworfen, deren Mitglieder sich alle an einem Ort befinden und die jeden Tag für Meetings zusammenkommen können. Es wird derzeit versucht Scrum auf verteilte Entwicklungsumgebungen anzupassen.