3. Software Lebens-Zyklus Modelle

Ein Software Produkt beginnt in der Regel als eine Idee, ein vages Etwas: "Wie wäre es, wenn wir mit dem Computer die Kaffee - Maschine steuern könnten?"

Nachdem irgend jemand bewiesen hat, dass die neue Software nötig ist, wird das Produkt entwickelt, der Kunde nimmt es in Betrieb und alle sind glücklich und zufrieden: der Software Entwickler, weil er sehr viel Geld verdient hat; der Benutzer, weil er mit der neuen Software extrem effizient arbeiten kann und sehr viel Geld durch rationelleres Arbeiten spart. Alle sind glücklich, nur wegen der Software.

Das wäre das eine Szenario.
Erfinden Sie ein anderes, aber sind Sie nicht zu hart mit den Benutzern und den Software-Ingenieuren!

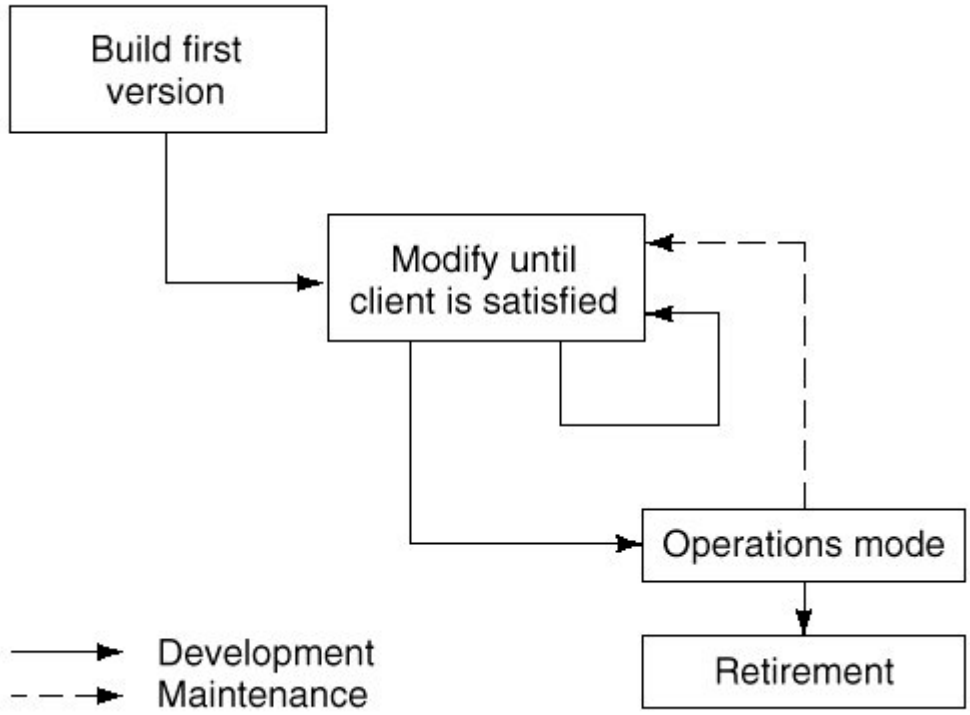
Die einzelnen Phasen, welche die Software durchläuft, von der Idee bis zum "verschrotten", bezeichnet man als *Lebens-Zyklus-Modell.*

Der Lebens-Zyklus jedes Produktes ist unterschiedlich! Für einzelne Produkte wendet man sehr viel Zeit auf zur Entwicklung des Konzeptes. Die Realisierung von Serienprodukten darf in der Regel nicht lange dauern. Auch bei Software, die zum Beispiel für ad hoc Auswertungen eingesetzt werden soll, wird kaum sehr lange am Leben bleiben: die Benutzerwünsche ändern sich zu schnell!

3.1. Build-and-Fix Modell

Die meisten Produkte, speziell im Software- Umfeld, werden mit der Kenntnis entwickelt, dass sie schon bald einmal wieder geändert werden müssen.

Der adäquate Ausdruck für ein solches Vorgehensmodell ist "Build-and-Fix".
Schematisch sieht das Modell wie folgt aus:



Diese Art der Programm-Entwicklung ist brauchbar, sofern das Programm einfach ist und sich ein eigentliches Projektmanagement nicht lohnt, also für Projekte mit 200-400 Zeilen Code typischerweise!

Für komplexere Projekte bezw. Produkte verbietet sich dieses Vorgehen von selbst, da viel zuviel Geld auf dem Spiel steht. Das Risiko eines Scheitern des Projektes wäre viel zu gross!

Aber in kleinen Projekten sind auch die Änderungen entsprechend klein, das Risiko gering und der Zyklus entsprechend schnell!

Betrachtet man die Kosten, die entstehen, durch wiederholtes Umprogrammieren, so zeigt sich auch beim kleinsten Projekt, dass es wesentlich günstiger gewesen wäre, wenn man einen sauberen Plan gemacht hätte.

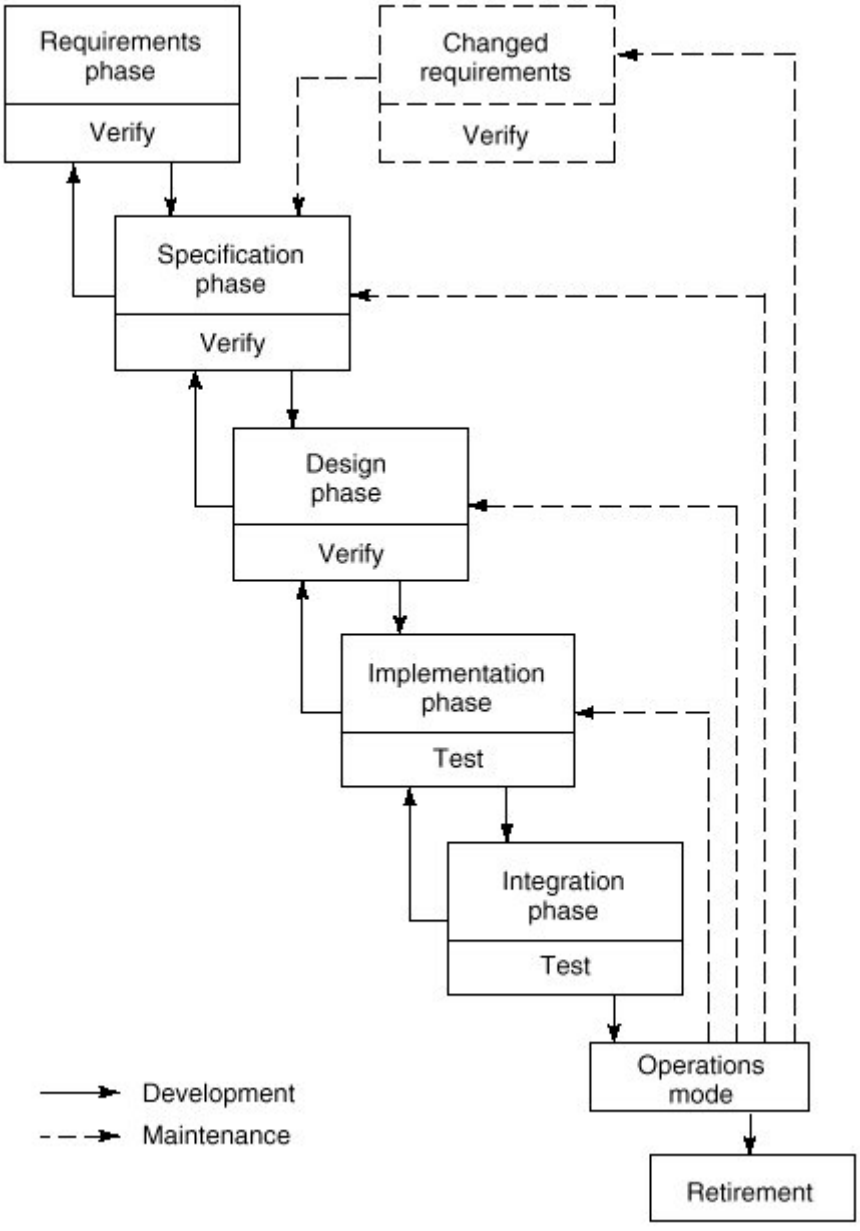
Dazu muss ein brauchbares *Modell* definiert werden.

Bei einem typischen Projekt wird als erstes ein "Game-Plan" definiert, ein Projektplan. Die Qualität des Projektplanes wirkt sich direkt auf Erfolg oder Misserfolg des Projektes aus!

Für viele Projekte hat sich bewährt Standard-Projektpläne und eine Standard-Vorgehensweise zu wählen. Alle grösseren Software-Häuser, staatliche Stellen, das Miltär und die Weltraum-Behörden sowie verschiedene Berufs-Organisationen (IEEE) aber auch Normierungsbehörden, haben ihr Vorgehensmodell veröffentlicht.

Nachdem man sich auf ein Vorgehensmodell eingeschossen hat, muss der Projektleiter die einzelnen Phasen des Modells weiter verfeinern. Zudem muss ein Reporting-System eingeführt werden, damit jederzeit klar ist, was bis wann von wem erledigt werden muss bezw erledigt wurde.

Das Wasserfall-Modell



Das Wasserfall Modell stammt (vermutlich) aus dem Jahre 1970, ist also noch nicht sehr alt, aber punktuell sicher überholt. Für viele ist es immer noch Stand der Technik, also befassen wir uns damit.
Zum Ablauf:
der Software Qualität Sicherung geprüft und vom Kunden

3.1.1. Anforderungsphase

Als erstes werden die Anforderungen an das zu erstellende System erfasst und schriftlich festgehalten.
Diese sind erste Ideen, wie das System aussehen könnte und was das System können muss.

3.1.2. Spezifikationsphase

Die Spezifikation basiert auf den Anforderungen und hält diese schriftlich fest.

Diese Spezifikation wird sowohl vom Kunden als auch vom Entwickler unterschrieben und von der Software Qualität Sicherung überprüft.

Die Spezifikation beschriebt das WAS, **nicht das WIE!** Auch dieses Dokument wird geprüft und vom Kunden unterschrieben.

3.1.3. Designphase

Jetzt beginnt im Sinne des Software Engineerings die Planung: ein detaillierter Zeitplan wird erstellt, alle Meilensteine werden fixiert und die vorhandenen Resourcen werden überprüft (Human Resources, Budget, Zeit). Diese Planung wird von der Software Qualitäs-Sicherung, dem Kunden und dem Entwickler geprüft und unterschrieben.

Nun kann der eigentliche Entwicklungsprozess beginnen.
Beim System Design geht es primär darum, die Systemgrenzen aufzuzeichnen und eine erste Aufteilung in überschaubare Blöcke zu erreichen!

Im Design Dokument wird das **WIE** beschrieben.

Unter Umständen , also eigentlich in der Regel, zeigen sich in der Design- Phase Probleme , Lücken oder Fehler in der Spezifikation. Der Grund dafür ist unter anderem der Detaillierungsgrad der einzelnen Phasen:

In der Anforderungsphase werden die Anforderungen aus Benutzersicht (das WAS) festgehalten; diese sind zwangsweise nicht so präzise wie ein Computer Programm

Wesentliche Teile, die in der Spezifikation fehlen, müssen jedoch nachgeholt werden! Wir müssen also de facto zurück in die Anforderungsphase!

Als nächstes müssen nun die Design-Dokumente nach geführt werden und die Planung gegebenenfalls revidiert werden. Dieses iterative Vorgehen kann sich mehrfach wiederholen, sollte jedoch in der Regel nicht mehr als drei mal durchlaufen werden; man kann ein System auch überspezifizieren.

Das Design-Dokument ist damit abgeschlossen und wird an den Programmierer, den Implementierer abgegeben.
Oft unterscheidet man verschiedene Ebenen des Designs:

- Architektur-Design : in ihm werden die wesentlichen Strukturen festgelegt

- Modul-Design : in ihm wird festgelegt, wie die Architektur implementiert werden soll

3.1.4. Implementationsphase

Wie in jeder Phase wird sich auch bei der Implementierung zeigen, dass einzelne Designetscheide nicht korrekt waren. Aus diesem Grund muss man die Möglichkeit haben, wieder zurück in die Designphase zu wechseln und natürlich die entsprechenden Dokumente nach zuführen.

Im schlimmsten Fall muss sogar weiter als in die Designphase zurück gegangen werden ("backtracking" im übelsten Sinne).

3.1.5. Integrationsphase

Der Entwickler hat den Eindruck seine Arbeit wäre fertig und der Integrator versucht daraus das Beste zu machen. Die einzelnen Module, Objekte, was auch immer, werden zum fertigen System zusammen gebaut.
Die Funktionsfähigkeit des Systems kann anhand der bereits vorliegenden Dokumentationen überprüft werden.

Öfters als wünschenswert zeigen sich verschiedene Probleme und die Schnittstellen, einzelne Module müssen geändert werden ,oder weitere Module müssen zusätzlich erstellt werden..

Die Integrationsphase schliesst mit einem protokollierten Integrationstest.

Es folgt ein Abnahmetest durch den Auftraggeber beziehungsweise den Benutzer. Je nach Komplexität des Projektes kann man verschiedene Vorversionen freigeben:

- Die Alpha Version, die im wesentlichen verifiziert, dass das Ergebnis den Spezifikationen entspricht
- Die Beta Version, die einer grösseren Anzahl Benutzer zur Verfügung gestellt wird, bevor die Software in den Massenmarkt freigegeben wird

3.1.6. Operationsphase inklusive Wartung

Der Benutzer hat das Ergebnis (zähneknirschend) akzeptiert und versucht das Produkt einzusetzen: die Nutzungsphase der Software hat begonnen.

Im Gegensatz zur Hardware beginnt man auch bereits intensiv das Produkt zu erweitern und Fehler zu eliminieren. Das ist etwa vergleichbar mit einem Auto, welches am Tage nach dem Kauf bereits neu gestrichen wird, ein anderes Heck bekommt und auch noch fliegen lernt!

3.1.7. Zusammenfassung

Kritisch beim Wasserfall Modell ist, dass jede Phase mit einem schriftlichen Dokument abschliesst. Dieses Dokument ist abgesichert, durch die Software Qualitätssicherung und den Benutzer. Dadurch will man vermeiden, dass zu viele Änderungen zu einem späten Zeitpunkt ins Projekt einfliessen.

Das Wasserfall Modell hat sich in der Praxis eigentlich sehr gut bewährt! Allerdings gab es wie immer auch einige Fehlschläge!

3.1.8. Analyse des Wasserfall Modells

Das Wasserfall hat viele Vorteile, unter anderem das disziplinierte Vorgehen. Die einzelnen Phasen müssten eigentlich sauber dokumentiert sein! Die Überprüfung der Ergebnisse durch mehrere Stellen führt zu einem hohen Grad an Vollständigkeit!

Inhärent im Wasserfall Modell ist das Testen: in jeder Phase finden eigentlich phasenspezifische Tests statt.
Testen ist keine eigene Phase: die Qualität soll ins Produkt hinein konstruiert werden, nicht nachträglich erreicht werden. Dieses Vorgehen lehnt sich an das Vorgehen in der Hardware Produkte Entwicklung an. Auch dort wird die Qualität nicht am Schluss ins Produkt geprüft.

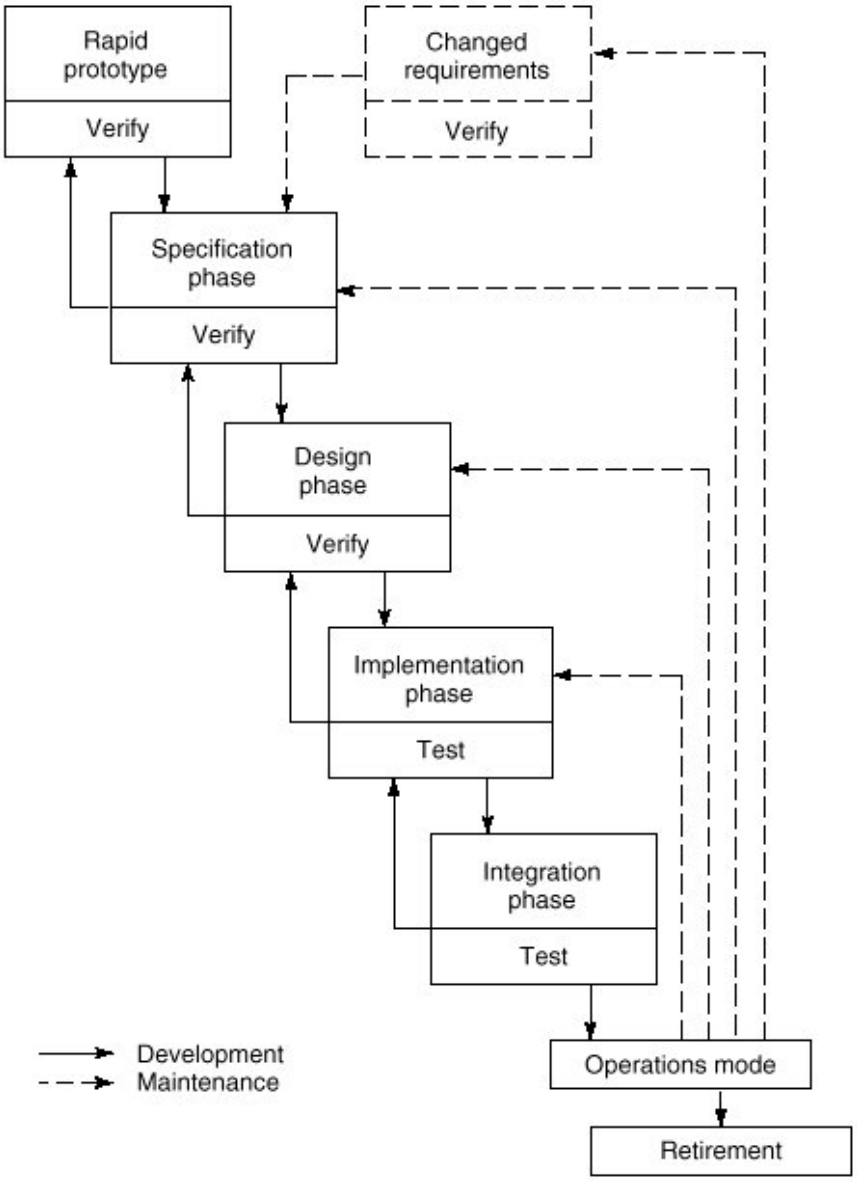
Der Vorteil des Wasserfall Modells, seine saubere Dokumentation, ist zugleich ein Nachteil des Wasserfall Modells.: falls ich ein Auto kaufen will, gebe ich mich auch nicht mit einer technischen Beschreibung des Fahrzeuges zufrieden; ich möchte das Auto sehen und unter Umständen testen.

3.2. Rapid Prototyping Modell

Ein *rapid prototype* ist ein funktionierendes Modell des zu erstellenden Systems mit der Einschränkung, dass lediglich bestimmte Funktionen implementiert wurden.

Beispiele:
Jemand möchte zum 8 Milliardesten Mal eine Buchhaltung programmieren, natürlich eurofähig, speziell für die Schweiz und in Romatsch!
Damit die Graubündner Wirtschaft-Förderung bezahlt, muss das Software-Haus innerhalb einer festgelegten Frist die Datenerfassungsmasken zeigen.

Bei einer Prozesssteuerung muss die Enzymkonzentration eines Fertigungsprozesses überwacht werden. Der Prototyp implementiert lediglich die Konzentrationserfassung, ohne Vaildation und Plausibilitätsprüfung.



Der erste Schritt beim Rapid Prototyping Modell ist das schnelle Erstellen eines Prototypen und das Experimentieren mit diesem.

Nachdem der Kunde damit zufrieden ist, wird die Spezifikation schriftlich festgehalten und der eigentliche Entwicklungsprozess kann beginnen.

Vermutlich werden weniger Feedback Loops benötigt als beim Wasserfall Modell, da die Abstimmung der Benutzerwünsche mit den Entwicklern bereits erfolgte.
Der Entwickler wurde bereits sehr früh direkt mit dem Benutzer in Kontakt gebracht!

3.2.1. Integration von Wasserfall und Rapid Prototyping Modell

Trotz den vielen Erfolgen des Wasserfall Modells zeigen sich auch klare Schwächen, die mit dem Rapid Prototyping Modell besser gelöst werden:

- Die frühe Auseinandersetzung des Benutzers mit dem Endprodukt, auch wenn es nur ein Prototyp ist, verbessert die Akzeptanz und vermeidet das Problem, dass der Benutzer mit einer Spezifikation des Produktes (auf Papier) sich wenig vorstellen kann.
- Da beim Wasserfall Modell im Extremfall in jeder Phase neue Personen involviert sind (ausser dem Benutzer) bestehen oft Kommunikations -Schwierigkeiten .

Eine Lösung besteht darin, die zwei Phasen Modell zu kombinieren:

- Der Prototyp wird als Input für die Produkte-Spezifikation verwendet
- Die Anwender Bedürfnisse lassen sich besser formulieren und erkennen
- Nach der Festlegung der Produkte Spezifikation wird gemäss dem Wasserfall Modell fortgefahren
- Der Prototyp zeigt auch allfällige Technologie Probleme auf und hilft, das Risiko eines Fehlschlages abzuschätzen.

Auf eine detailliertere Analyse des Rapid Prototyping Modells verzichten wir hier, da wir später nochmals darauf eingehen werden (in der detaillierten Analyse der Requirement Phase).

3.3. Inkrementelles Modell

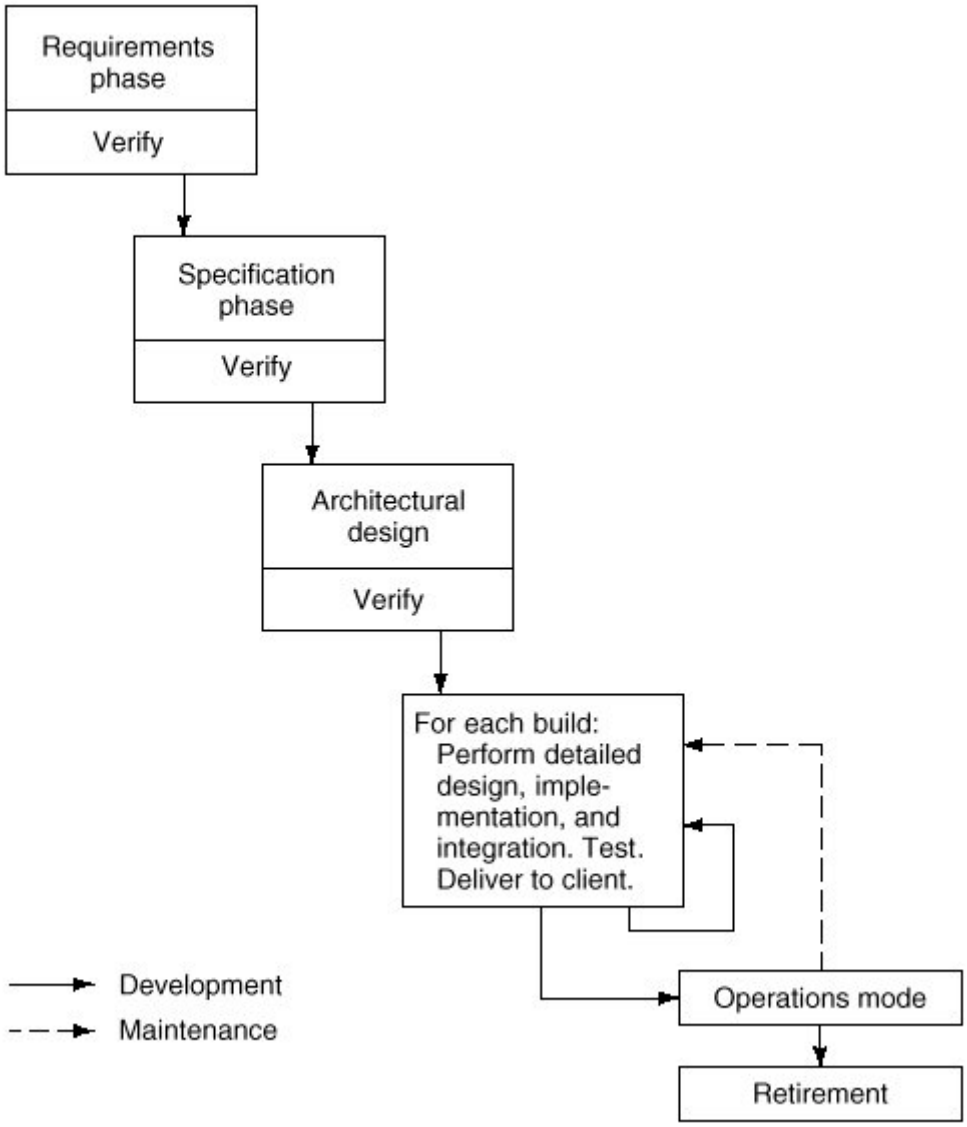
Software wird gebaut, nicht geschrieben. Deswegen ist sie auch nicht fertig nach der Auslieferung.
Die Software wächst also weiter, wird erweitert, ergänzt, verbessert und erneuert.

Das Wasserfall-Modell ist somit als Grundmodell zwar sinnvoll, aber in der Regel muss man von einem andern, realeren Vorgehen ausgehen.

Die Zeichnung unten zeigt, wie konkret vorgegangen wird:

Nach einer Anforderungs-, Spezifikations- und einer Architektur Design Phase wird beim Module Design und der Implementierung schrittweise vorgegangen.

Die Funktionalität wird somit schrittweise hinzugefügt. Der Benützer erhält sein gewünschtes System in Phasen.



3.3.1. Analyse des Inkrementellen Modells

Das Ziel jedes Lebens-Zyklus Modells ist das Erstellen und liefern von robuster, qualitativ hochstehender Software, im Budgetrahmen für Zeit und Geld erstellt, und gemäss den Anforderungen des Benutzers.

Das Wasserfall und das Prototyping Modell führen in der Regel zu einem System, welches gründlich getestet wurde, dokumentiert ist aber unter Umständen erst nach einer längeren Wartezeit dem Benutzer zur Verfügung steht.

Das inkrementelle Modell liefert dem Benutzer relativ schnell ein brauchbares Subsystem, welches er bereits produktiv einsetzen kann. Er erhält also bereits sehr früh einen Return für sein Investment.

Der Benutzer braucht auch keine Angst zu haben, dass er nach einer (viel zu) langen Wartezeit schliesslich ein Produkt erhält, welches

- Technologisch überholt ist
- Nicht mit dem zu tun hat, was er jetzt benötigt
- Sehr viel Geld gekostet hat aber eben kaum brauchbar ist

Die Schwierigkeit beim inkrementellen Modell liegt darin, dass jeder neu freigegebene Modul ins bereits gelieferte Software Paket passen muss.

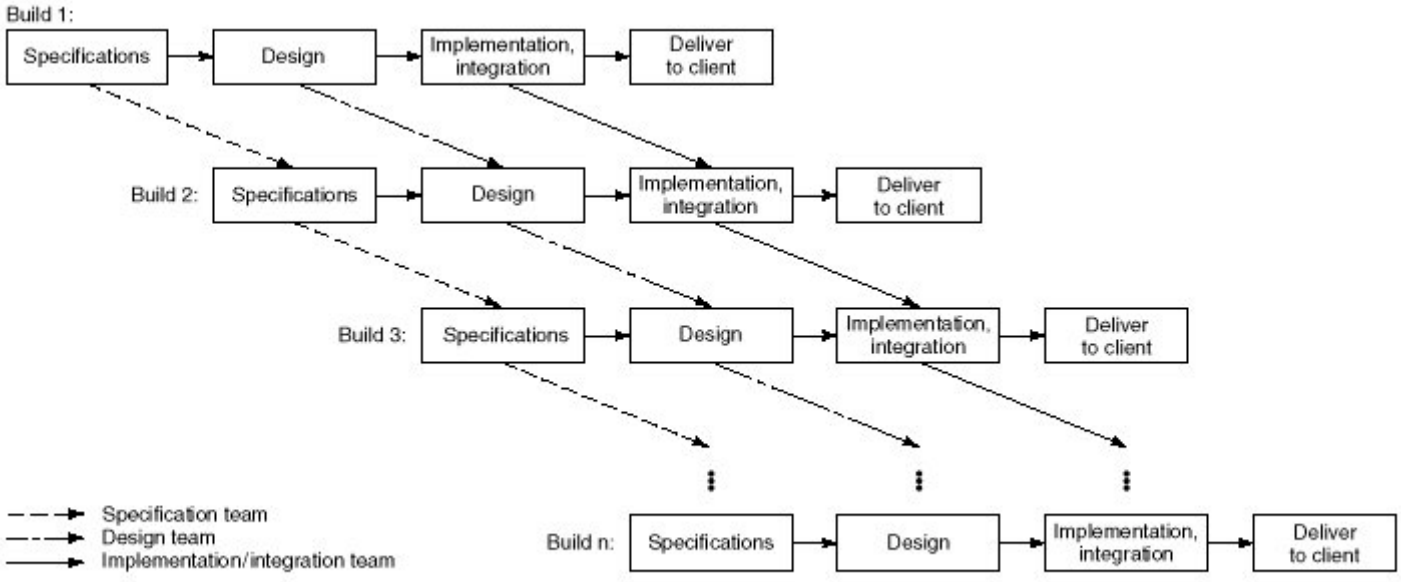
Die Architektur des Systems muss also von Anfang an erweiterbar ausgelegt worden sein.

Im schlimmsten Fall wurden einzelne Module durch Wartung, Erweiterungen und Ergänzungen so verändert, dass laufend sich ändernden Schnittstellen nach programmiert werden muss.

Zudem wird das Software Haus mehr und mehr Personal für den Unterhalt frei stellen müssen; die Einführung neuer Module verlangsamt sich somit zunehmend.

Da die Anforderungen sich laufend ändern können, degeneriert im schlimmsten Fall das Modell zum Build and Fix Modell.

Es gibt Varianten vom inkrementellen Modell. Eine davon sehen wir im Folgenden:



Diese Version ist gefährlich: die Architektur des Systems kann sich nach jeder neuen Spezifikation eines Teilsystems ändern.

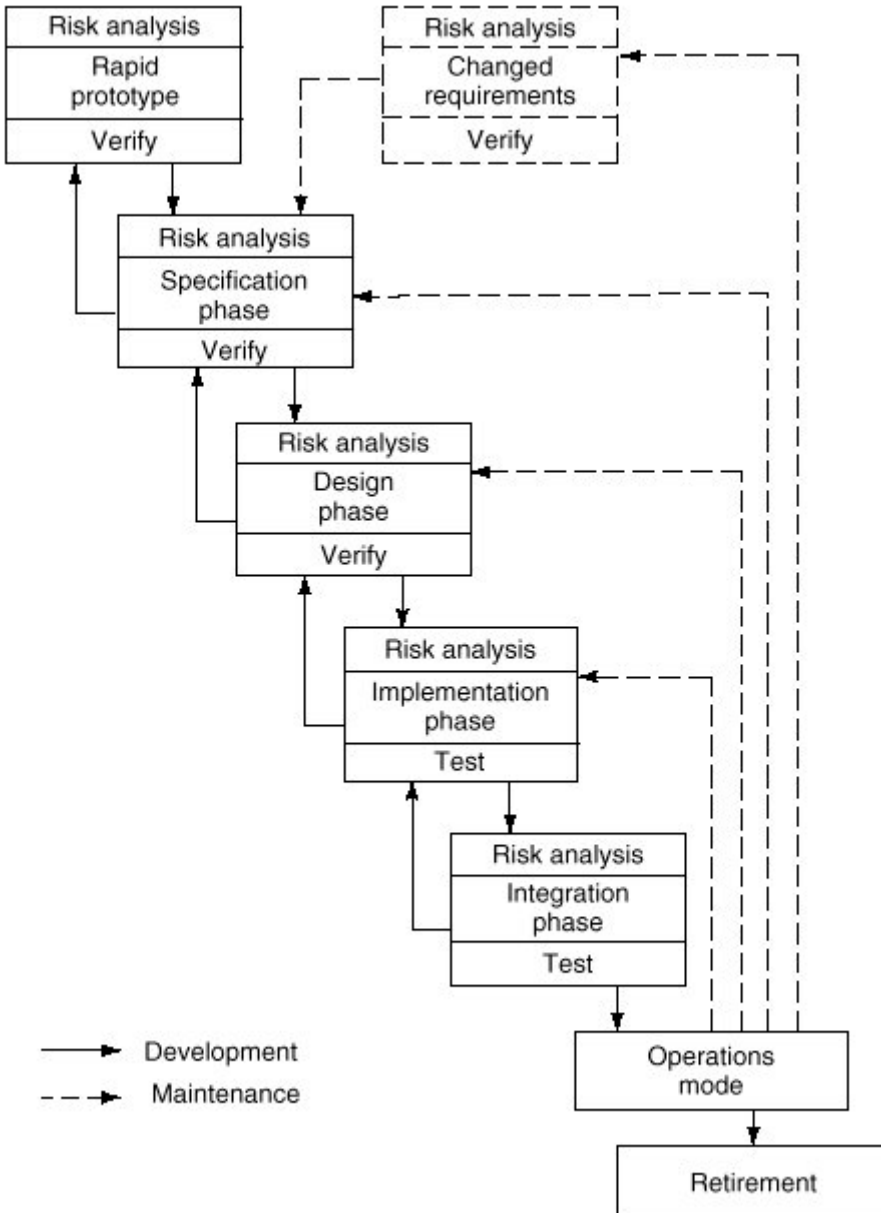
Architektur- Änderungen sind ein einem laufenden Entwicklungs- Zyklus sicher nach Möglichkeit zu vermeiden.

3.4. Spiral Modell

Jedes Modell hat seine Risiken und seine Vorteile. Im schlimmsten Fall laufen die Schlüsselpersonen davon, der Hardware Lieferant verspricht zu viel, die Basissoftware (Betriebssystem, Kommunikations- Software, Datenbank- Software) ist instabil und führt zu Frustration und unnötigen und unplanbaren Verzögerungen.

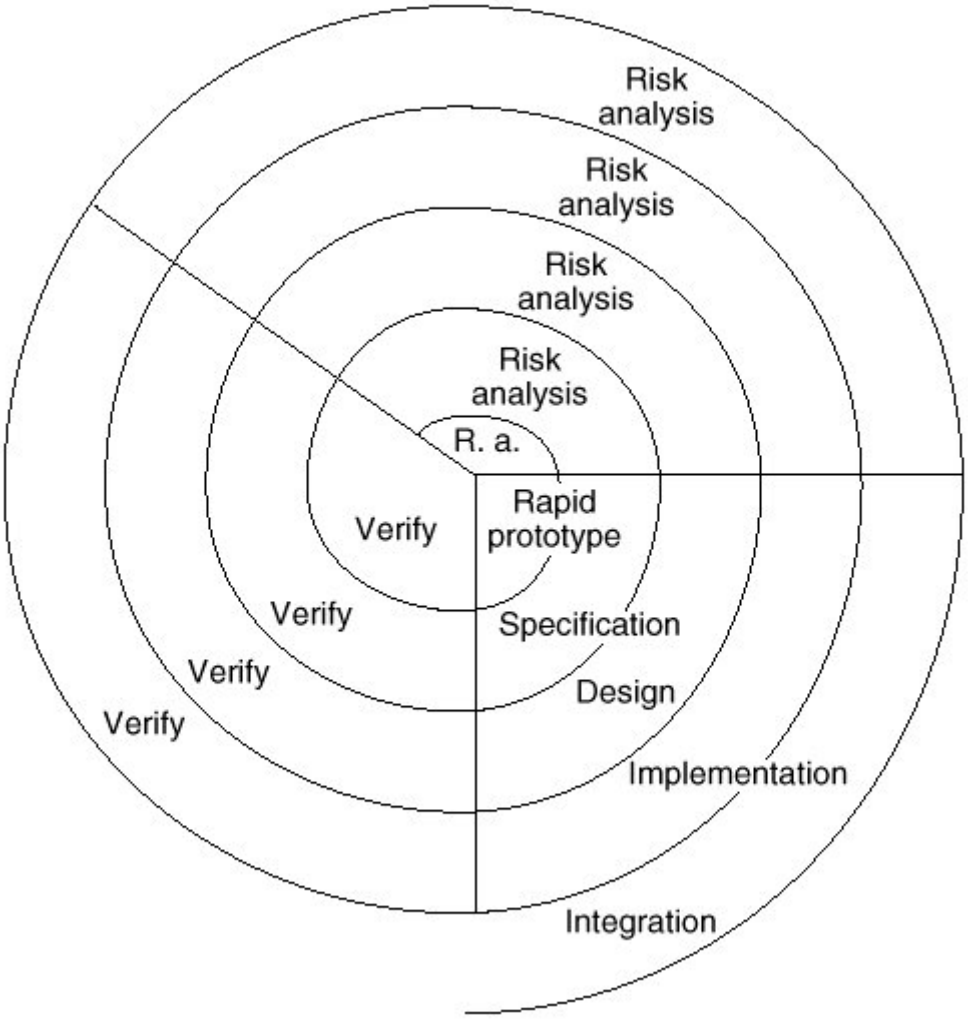
Die Grundidee der Risiko Abschätzung BEVOR die einzelnen Phasen beginnen, gefolgt von einer Validation NACHDEM die einzelnen Phasen abgeschlossen sind, führte zur Entwicklung des Spiralmodells.

Das Spiralmodell sieht, analog zu den andern Modellen dargestellt, wie folgt aus:

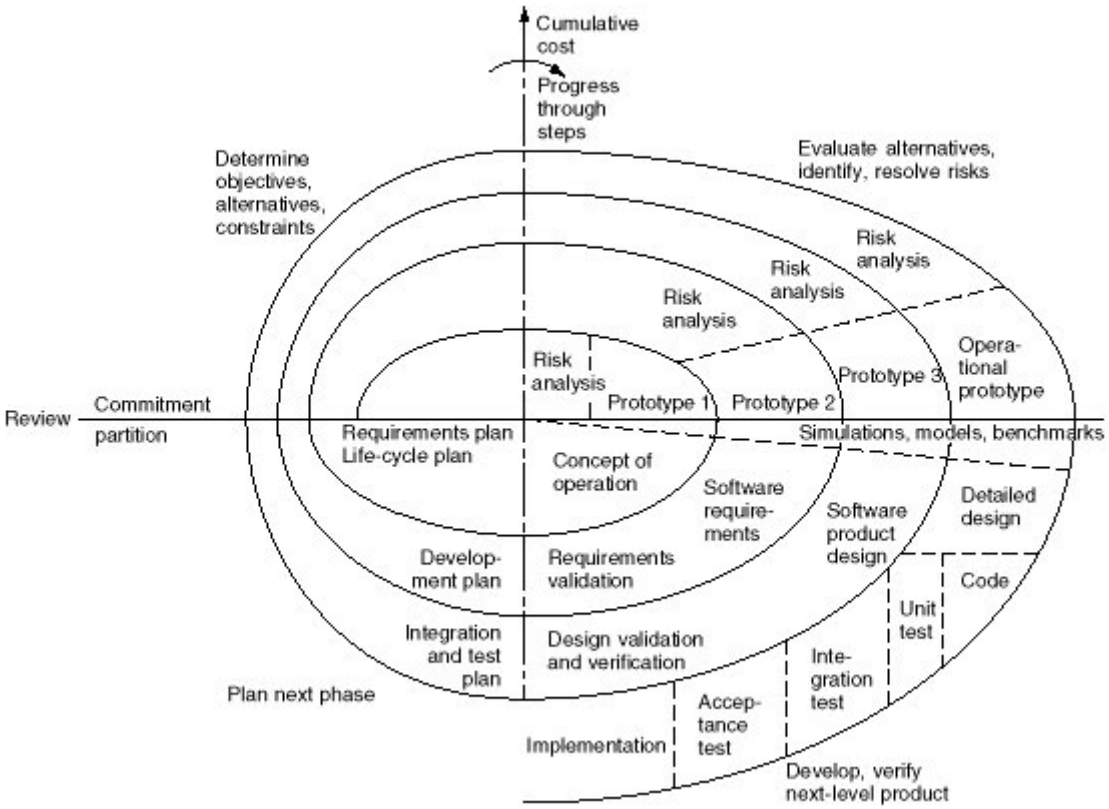


Das Spiralmodell entspricht also (vereinfacht gesagt) dem Wasserfall- Modell, allerdings mit einer zusätzlichen Risiko Analyse vor jeder Phase und einer Validation nach jeder Phase.

Die Spirale erkennt man besser, wenn man die Wartungsphasen (in denen ja auch Entwicklung geschieht) spiralförmig an die Operationsphase anhängt:



Wenn wir das Modell weiter verfeinern gelangen wir schliesslich zum Wasserfall Modell wie es von Boehm vorgestellt wurde, und im bereits verteilten Artikel beschrieben wird.



Im Wesentlichen sehen Sie einige zusätzliche Details, wie Test-Phasen, Planungs-Schritte etc.

Was sind denn typische Risiken in der Software Entwicklung?
Einige haben wir schon erwähnt:

- Eine Schlüsselperson kann aussteigen
- Der Kunde kann von einem andern Konzern übernommen werden, dann lohnt sich die Entwicklung nicht mehr.
- Bestimmte Teilfunktionen sind sehr schwierig zu realisieren ;
- die Technologie ist neu und noch unerprobt
- ...

3.4.1. Analyse des Spiralmodells

Das Spiralmodell hat zweifellos seine Stärken.
Die Risiko Analyse führt dazu, dass jederzeit klar was, was auf dem Spiel steht, für das Management und die Informatik.

Ein Problem ergibt sich in Bezug auf das Gesamtprojekt:
Wie ist eine Gesamtplanung möglich?
Wieviel Resourcen werden insgesamt benötigt?
Wann ist das Produkt fertig? (NIE!)

Einige dieser Fragen möchte das Management beantwortet haben, aber eine Antwort ist in der Regel äusserst schwierig.

Die Grösse des Projektes spielt ebenfalls eine wichtige Rolle:

Das Spiralmodell wird normalerweise nur bei Grossprojekten eingesetzt, da das Risiko in kleinen Projekten sicher nicht im Zentrum des Interesses steht.

3.5. Vergleich der Lebens-Zyklus Modelle

Wir haben fünf unterschiedliche Lebens-Zyklus Modelle vorgestellt. Jedes hat seine Stärken und Schwächen.

Das Wasserfall Modell hat sicher, speziell als universelles Modell, seine Stärken. Es hat sich auch in der Praxis bereits sehr bewährt. Seine Stärken und Schwächen konnten an Hand vieler Projekte untersucht werden

Das Rapid Prototyping Modell ist weniger gründlich untersucht worden. Es hat sich aber sicher immer dann bestens bewährt, wenn es darum geht, dem (ungeschulten) Benutzer einen Eindruck zu vermitteln von dem, was ihn am Schluss erwartet.

Das inkrementelle und das Spiralmodell spielen eine spezielle Rolle und sind definitiv nicht universell einsetzbar!

Sie werden später sehen, dass bei der Objekt Orientierten Entwicklung weitere Kriterien eine Rolle spielen, die zu Modifikationen der Basis-Modelle führen.

Jede Software Organisation muss sich ihr Modell selber zusammen stellen, basierend in der Regel auf einem der besprochenen Basismodelle.

Die konkrete Wahl eines bestimmten Modells hängt von vielen Faktoren ab, nicht zuletzt von den Personen, die am Projekt beteiligt sind.

3.6. SOFTWARE QUALITÄT

Da dies keines meiner Lieblingsthemen ist, ich mich aber einige Zeit damit befasst habe, möchte ich versuchen kurz einige wesentliche Punkte zu erwähnen, die heute üblicherweise unter Software Qualität geführt werden.

Grundsätzlich stellt sich die Frage, wie man Qualität messen soll:

- Ist ein Produkt qualitativ hochstehend, wenn es technisch hochstehend ist, aber der Benutzer damit nichts anfangen kann?
- Ist ein Produkt qualitativ hochstehend, wenn der Benutzer es gut findet (aber vielleicht nur 10% einsetzt), aber die Wartung des Produktes kaum möglich ist?

Neben diesen Extremfällen gibt es ein grosses Spektrum möglicher "Qualitätsmerkmale".

Das Software Engineering Institute (SEI) an der Carnegie Mellon University (CMU), einer DER Spitzenuniversitäten im Informatikbereich, wurde ein Modell entwickelt, nicht wie Software entwickelt werden soll, sondern wie der Software Entwicklungs Prozess verbessert werden kann.

Dieses Modell, das Capability Maturity Model (CMM) möchten wir kurz besprechen.

Das Ziel des Software Entwicklungs-Prozesses ist die Herstellung von Qualitätssoftware im Rahmen eines vordefinierten Budgets und eines sauberen Zeitplanes.

Der Software Entwicklungsprozess besteht aus Management- und Entwicklungs-Aufgaben und ist sicher wesentlich anspruchsvoller als die meisten andern Entwicklungsprojekte in anderen Domänen. Aber: "Informatik Entwickler sind ja auch besser als die meisten andern Entwickler".

CMM geht von einem inkrementellen Prozess aus: das Ziel, bessere Software besser zu entwickeln, lässt sich nur schrittweise, in Stufen entwickeln.

CMM definiert fünf Level:

	Maturity Level	**Charaktreisierung**
1	Initial	Ad hoc Prozess
2	Repeatable	Grundlegende Projektmanagement Methoden
3	Defined	Prozess definiert
4	Managed	Prozess überwacht
5	Optimized	Prozess kontrolliert

3.6.1. Maturity Level 1 : Initial Level

Auf der tiefsten Stufe macht jeder Software Entwickler was immer er will. Die Konsequenzen sind: nicht nachvollziehbare Software Entwicklung.

Der Software Entwicklungsprozess auf Stufe 1 ist ad hoc und personenabhängig. Ob ein Projekt erfolgreich oder nicht erfolgreich abgewickelt werden kann, hängt von den Personen, nicht vom (nicht existierenden) Prozess ab.

Zeit und Kosten Budgets werden eher zufällig eingehalten, wenn überhaupt.

Viele Organisationen befinden sich auf dieser Stufe, eine Schande!!

3.6.2. Maturity Level 2 : Repeatable Level

Auf diesem Level sind die grundlegenden Software Engineering Methoden eingeführt. Planung und Management Techniken basieren auf Erfahrungswerten aus vergangenen Projekten, von daher auch die Bezeichnung *"repeatable"*.

Der Software Prozess wird "gemessen" dh es werden Metriken eingeführt. Unter anderem werden Kosten und Aufwand verfolgt. Probleme werden vom Management erkannt und korrektive Massnahmen eingeleitet.

Wesentlich ist das Einführen von Metriken: dadurch wird der Prozess MESSBAR. Die Messdaten können in Folgeprojekten verwendet werden und sollten zu einer Verbesserung des Gesamtprozesses, speziell der Planung führen.

3.6.3. Maturity Level 3 : Defined Level

Auf Stufe 3 ist der Software Prozess definiert und dokumentiert. Management und technische Aspekte des Prozesses sind klar definiert und das Management ist bemüht, die Prozesse zu verbessern.

Mit Hilfe von Reviews wird die Software Qualität überwacht.

CASE Tools werden zur Verbesserung der Dokumentation und der Reproduzierbarkeit eingeführt. Solche Tools würden auf Level 1 das Chaos noch verstärken. Eine Studie einer Universität in Belgien hat gezeigt, dass zusätzliche Investitionen auf Level 1 zu einem Produktivitätsverlust führen; man gibt also auf Level 1 viel Geld aus, erhält aber nicht dafür. Auf Level 2, mit definierten Prozessen, wirken sich Investitionen in neue Technologien positiv aus.

Level 3 ist der heute von den BESTEN Organisationen erreichte Level. Deutsche Grosskonzerne sind bestenfalls auf Level 2.3 (also Level 2).

3.6.4. Maturity Level 4 : Managed

Auf Level 4 werden für jedes Projekt Qualitäts- und Produktivitäts- Ziele gesetzt und verfolgt. Diese zwei Prozessgrössen werden periodisch verfolgt und für statistische Zwecke erfasst und ausgewertet.

3.6.5. Maturity Level 5 : Optimized

Auf Level 5 bemüht sich eine Organisation intensiv und dauernd ihren Software Entwicklungs- und Management- Prozess zu verbessern.

Statistische Qualitätskontrolle wird durch gehend eingesetzt und dient der Optimierung der Prozesse und der Organisation.

Die Ergebnisse eines Projektes werden systematisch in neue Projekte einfliessen und zu einer kontinuierlichen Verbesserung führen.

3.6.6. Zusammenfassung

Die fünf Prozesse sind in der Tabelle zusammen gefasst. Normalerweise muss eine Organisation diese Leiter stufenweise "erklimmen". Als erstes muss also der aktuell implementierte Prozess verstanden werden, dann müssen systematisch Qualitätsdaten und weitere Prozessdaten erfasst werden (es muss ein Metriksystem eingeführt werden). Auf den oberen Stufen ist es wesentlich, dass DAUERND versucht wird, Verbesserungen zu erzielen,

Trends zu beobachten, neue Techniken zu evaluieren, zu erproben und kontrolliert einzuführen.

3.6.7. Erfahrungswerte

Um von Level 1 auf Level 2 zu gelangen benötigt eine durchschnittliche Organisation 3-5 Jahre. Das zeigt wie schwierig es ist, systematisch Software zu entwickeln.

Gemäss den "Erschaffern" des CMM Modells sollte eine Organisation den Schritt von einem Level auf den nächsten in 18 Monaten bis 3 Jahren schaffen, falls sie sich intensiv darum bemüht.

Pro CMM Stufe wurden bestimmte Schlüssel-Prozesse definiert, die von der Organisation speziell beachtet und verbessert werden sollten, um die nächste Stufe erreichen zu können.

Für jeder Stufe sind auch Ziele definiert, die erreicht werden sollten, um auf die nächste Stufe zu gelangen.

3.6.7.1. Beispiel:

Auf Stufe 2 ist einer der Schlüsselprozesse die Qualitätssicherung, Projektmanagement und Konfigurationsmanagement. Ziel des Projektmanagements ist es zu einer besseren und genaueren Planung zu gelangen.

Auf Stufe 5 geht es um präventive Massnahmen zur Vermeidung von Problemen, Technologie-Innovation und Prozess-Management.

Der Unterschied zwischen Level 2 und Level 5 ist also unter anderem, von der Qualitätskontrolle zu einem präventiven Prozessmanagement.

Um den Organisationen beim Verbesserungs-Prozess behilflich zu sein, hat das SEI Checklisten herausgegeben. Diese bilden auch die Basis für den Software Prozess Audit durch das SEI, falls man offiziell auf eine der Stufen eingeordnet werden möchte (oder muss, falls man für den MIL oder den Space Bereich Software entwickelt).

SEI wird im wesentlichen durch das DOD (Departement of Defense) bezahlt. Hughes Aircraft hat für die Verbesserung des Software Prozesses schon über 500'000 USD ausgegeben, es geht also um viel Geld. Hughes ist jetzt auf Level 3, und rechnet damit, dass durch die Verbesserungen jährlich ungefähr 2'000'000 USD gespart werden.

Vergleichbare Ergebnisse werden von andern Firmen berichtet: Raytheon (Atom U-Boote, Elektronik, ...) gelang von 1988 (Level 1) auf Level 3 (1993). Gemäss Raytheon führt jeder investierte USD zu einem Return von 7.7 USD, ist also gut investiert.

Die folgende Tabelle zeigt projizierte Kosten, Dauer, Aufwand pro Stufe:

CMM Level	Dauer (Kalender Monate)	Aufwand (Personen Monate)	Gefundene Fehler pro Entwicklung	Ausgelieferte und installierte Fehler beim Kunden	Total Entwicklungs-Kosten
Level 1	29.8	593.5	1'348	61	5'440'000 USD
Level 2	18.5	143	328	12	1'311'000 USD
Level 3	15.2	79.5	182	7	728'000 USD
Level 4	12.5	42.8	97	5	392'000 USD
Level 5	9.0	16	37	1	146'000 USD

Wie liest man die Tabelle?
Falls ein Projekt auf Stufe 1 29.8 Kalender Monate dauert, dann würde das gleiche Projekt bei einer Firma auf Stufe 5 lediglich 9.0 Kalender Monate dauern und auch nur einen Bruchteil kosten.

Die US Air Force verlangt bereits seit 1988 von ihren Lieferanten Stufe 3. Vermutlich wird auch das gesamte DoD in absehbarer Zeit ähnliche Forderungen an die Software Lieferanten stellen.

3.7. ISO 9000

CMM ist ein "Maturity" Prozess: man versucht den Software Prozess zu verbessern. ISO 900x hat ein ähnliches Ziel, unterscheidet sich jedoch deutlich von CMM.

ISO900x besteht aus 5 (vielleicht auch schon mehr) Standards, die eigentlich universell auf Industrie- und Wirtschafts- Prozesse angewandt werden können. Sogar Schulen können zertifiziert werden, sofern sie immer gleich (schlecht oder gut) sind.

ISO9000 ist also eigentlich kein typischer Software Prozess. Innerhalb der ISO 900x Standards ist ISO 9001 der am besten anwendbare Standard, obschon gemäss ISO der Standard 9000-3 (1991) eigentlich für den Software Prozess entwickelt wurde.

ISO betont speziell die Dokumentation. Prozesse müssen dokumentiert sein und immer gemäss dieser Dokumentation ablaufen (also immer gleich schlecht sein). Qualität ist nicht das eigentliche Ziel, obschon man das denken sollte. **Eine Firma auf CMM Stufe 1 kann durchaus ISO 9000 zertifiziert sein!**

ISO 9000 verlangt auch einen Messprozess, in wesentlich geringerem Masse als CMM. Der Vorteil von ISO im Gegensatz zu CMM ist die Verbreitung: ISO Standards werden in mehr als 90 Ländern eingesetzt. Viele Regierungsstellen und speziell die EU Stellen verlangen heute bereits die ISO Zertifizierung von ihren Lieferanten, ob Hardware oder Software spielt keine Rolle.

3.8. Zusammenfassung

Wir haben einige Prozesse beschrieben, die in der Software Entwicklung eingesetzt werden. Die Stärke des Wasserfall Modells ist das disziplinierte Vorgehen. Das Problem mit dem Wasserfall Modell ist, dass unter Umständen ein Produkt entsteht, welches der Kunde eigentlich nicht so haben möchte. Dieses Problem wird vom Rapid Prototyping angegangen. Das inkrementelle Modell, welches in ähnlicher Form speziell im OO Bereich eingesetzt wird, verbindet den Entwicklungsprozess mit der Wartung und setzt sich zunehmend durch, allerdings nicht generell.

ISO 9000 und CMM beschreiben nicht den Entwicklungsprozess; sie stehen quasi neben dem Prozess und achten darauf, dass alles optimal läuft, um qualitativ brauchbare Produkte abliefern zu können.

3.9. Selbtstestaufgaben

1. Sie sollen einen Algorithmus entwerfen, mit dessen Hilfe die Quadratwurzel aus 73'980'567 auf 4 Dezimalstellen nach dem Koma berechnet werden kann. Ihnen ist bekannt, dass das Programm danach nicht mehr benötigt wird. Welches Lebenszyklus-Modell würden Sie einsetzen?
2. Sie sind Software Engineering Berater und wurden von einem Finanzchef angerufen. Es geht um die Entwicklung eines neuen Software Systems, welches das Lager, den Einkauf und den Vertrieb automatisieren. Für die Firma stellt das System eine grössere Investition dar und das System soll entsprechend lange nutzbar sein. Welches Lebenszyklus Modell schlagen Sie vor und warum?
3. Geben Sie eine Liste mit 5 Entwicklungsrisiken für das Projekt aus Aufgabe 2. Wie kann man jedes dieser Risiken [einzeln] minimieren?
4. Sie waren im Projekt aus Aufgabe 2 erfolgreich. Die Branche ist an einer Lösung für viele Betriebe interessiert. Sie möchte aber, dass Sie das Produkt neu entwickeln und noch mit zusätzlichen Funktionen versehen, neben einer neuen Benutzeroberfläche. Das neue Produkt muss portabel sein, da die Branche heterogen ist dh. es gibt sehr grosse Betriebe, daneben aber auch kleine Firmen dh. Sie müssen mit unterschiedlichen Hardware und Systemsoftware Umgebungen rechnen. Inwiefern unterscheiden sich die Selektionskriterien für das Lebenszyklus Modells von jenen aus Aufgabe 2?
5. Welche Art Produkte lassen sich am Besten mit Hilfe des inkrementellen Modells entwickeln?
6. Wann führt das inkrementelle Modell fast sicher zu Schwierigkeiten?
7. Beschreiben Sie Software Produkte, die man idealerweise mit dem Spiralmodell entwickelt.
8. Wann ist das Spiralmodell sicher das falsche Lebenszyklus Modell?
9. Sie haben viel Geld verdient und beschliessen, die Firma Betakus zu kaufen. Nach dem Kauf stellen Sie fest, dass die Firma offensichtlich auf CMM Level 1 ist, also chaotisch organisiert ist und im wesentlichen vom Know How eines Software Gurus lebt. Was müssen Sie als erstes unternehmen?
10. Welches Lebenszyklus Modell würden Sie für die Entwicklung der Software der Case Study einsetzen?
11. Lesen Sie den Artikel von Boehm 1988 über das Spiralmodell.
    Zeigen Sie, dass das Spiralmodell sowohl Aspekte des Wasserfall Modells und des Rapid Prototypings berücksichtigt.
    Welches der beiden Modelle zeigt sich dominant im Spiralmodell?

Literatur Referenz:

B. W. Boehm : "A Spiral Model of Software Edevelopment and Enhancement"
IEEE Computer **21** (May 1988) pp 61-72