

Software Engineering Management

1. Ein Beispiel für ein mißlungenes Projekt (Corporate Information System)

1.1 Ein Softwareprojektdesaster

Ein Projekt mißlang, obwohl die Verantwortlichen alle richtigen Dinge getan haben.
So haben sie:

- verschiedene Management Publications zu Hilfe gezogen und die Idee eines zentralisierten, unternehmensweiten Informationssystems akzeptiert und übernommen
- es wurde ein sogenannter 'think tank' eingestellt, der eine Durchführbarkeitsstudie anfertigte, die zwei Kalenderjahre (15 Arbeitsjahre) bis zur Fertigstellung benötigte
- das Unternehmen verwendet die modernsten Computer und Datenbanksoftware
- es wurden ohne Protest mehrere Millionen Dollar an zusätzlichen Entwicklungskosten gezahlt, obwohl das Budget schon aufgebraucht war
- es wurden die neuesten strukturierten Programmtechniken verwendet

1.2 Aufgetretene Fehler

- Programm funktionierte - Programmierer und Projektmitarbeiter brauchten mehr Zeit um das System zu tunen - zu langsam
- 20,000 Transaktionen jeden Tag um ihn auf den aktuellen Stand zu bringen - eine Transaktion kann bis zu 20 Minuten dauern - Programm versagte, alle notwendigen Updates eines Tages auch an einem Tag durchzuführen
- 2 Jahre um eine neue Fabrik einzubinden - aber eine neue Fabrik alle 6 Monate

1.3 Konsequenzen

- Projekt mußte aufgegeben werden
 - gegenüber Konkurrenz 5 Jahre verloren
- sollte nicht passieren, tut es aber immer wieder

1.4 Das Desaster vermeiden

Systementwickler hatten nicht den Mut ihre Fehler zuzugeben - "Ich wußte das es nicht funktionieren wird, aber *meine* Programme haben funktioniert - keine Sorgen
Der Grund für das Versagen des Projekts: es wurden keine Project Attributes, die für das überleben des Programms wichtig waren, bestimmt und auch nicht kontrolliert. Es konnte daher auch kein, auf diese Attributes aufbauender, passender Aufbau des Programms erfolgen.
Kleinere Schritte

Critical Attributes: Qualität und Grundlagen - Kollaps eines Systems verursachen - unter bestimmten Limits (worst acceptable level)

CIS Fall: tägliche Transaktionsleistung nicht ausreichend und konnte auch nicht ausreichend verbessert werden - Integration neuer Firmen dauert zu lange - hätte Anzahl der Transaktionen pro Tag bestimmt werden müssen - 86 400 Sekunden in 24 Stunden → 1 Sekunde pro Transaktion

1.5 Genauere Leistungsangaben für CIS

Work Capacity: es muß möglich sein, die Arbeit eines normalen Tages an einem Bürotag zu erledigen

Worst Case: 4 Sekunden für eine durchschnittliche Transaktion

Planned level(in der Anfangsphase): <1 Sekunde pro durchschnittlicher Transaktion

Planned level(bei mehr als 100 000 Tr./Tag): 0.2 Sekunden pro Transaktion

Kein Zeichen davon in der Praxis - niemand hat sich Sorgen darum gemacht - genauso bei der Einbindung einer neuen Firma

1.6 Einbindungsspezifikationen für CIS

Adaptability: das System sollte in der Lage sein, eine neue Firma in einer Zeit integrieren können, so das das System selber nicht de bremsende Faktor ist

Worst acceptable case: nicht mehr als 6 Monate mit nicht mehr als 10 Programmierern

Planned level: wenig als 6 Arbeitsmonate mit gerechtfertigtem Aufwand

1.7 Architecture and Engineering

Da keine klar definierten Attribute requirement specifications gefunden wurden, fehlten diese auch im Engineering and architectural process.

z.B.: Arbeitskapazität war nicht klar definiert - große Datenbank vom Hardwarehersteller zur Verfügung gestellt zu langsam und ungeschickt - schwerwiegende Architecture decision - der Hersteller wollte nur Geld verdienen, Rest egal - alle diese fehlenden Specifications hätten aber in die Designphase der File und Programmorganisation eingebunden werden müssen, um die benötigten Ergebnisse zu erreichen. Die Aufmerksamkeit konzentrierte sich auf die Funktionen der Applikation und der Programmierung. - kritischen Faktoren wurden überlassen

1.8 Evolutionary delivery (evol. Entwicklung)

Alle diese Fehler hätten früh genug erkannt werden können, wenn man früh genug Erfahrung mit der Kapazität und Adaptierbarkeit gemacht hätte. Es wurde aber nicht Stück für Stück entwickelt, sondern alles war auf einen Termin ausgelegt an dem alles funktionieren sollte. Das heißt 5 Jahre Arbeit und es läuft, oder gar nichts.

Evolutionary delivery hätte das böse Erwachen verhindern können. Nach einer ersten Entwicklungsphase hätte festgestellt werden können, ob es sich auszahlt weiter zu machen und ob diese erste Phase überhaupt funktioniert.

1.9 Die CIS Fehler kommen in den meisten Software Engineering Projekten vor

Volvo Schweden

- **unklare Systemspezifikationen**
- **zu wenig Arbeit und ausgebildetes Personal (software architects & sw-engineers), um das System auf die wichtigen Attributes auszurichten - Programmierer durften den Aufbau des Programms gestalten**
- **kein Feedback und keine Chance aus Fehlern zu lernen - kein Evolutionary delivery - alles auf einen Termin ausgerichtet**

1.10 Grundlegende Prinzipien des Software Engineering Management

- **Das “unsichtbare Ziel” - Prinzip**
Alle kritischen Systemeigenschaften müssen klar bestimmt werden. Unsichtbare Ziele sind für gewöhnlich schwer zu treffen außer per Zufall
- **Das “alle-Löcher-im-Boot” - Prinzip (wenn nicht alle zu sind, geht es unter)**
Die Designlösungen müssen alle kritischen Eigenschaften auf einmal erfüllen
- **Das “den Nebel vom Ziel vertreiben” - Prinzip**
Alle kritischen Eigenschaften können in meßbaren, testbaren Einheiten definiert werden

- **Das “lerne bevor dein Budget verbraucht ist” - Prinzip**
Versuche niemals, ein großes und komplexes System auf einmal abzuliefern. Liefere immer kleinere Teile des Programms, damit Probleme früh erkannt und korrigiert werden können.
- **Das “fühle dich sicher” Minimisationsprinzip**
Wenn du nicht weißt was du tust, dann tue es nicht im großen Rahmen

2. Was ist das wirkliche Problem beim Managen eines Projekts?

2.1 Fehlen der Klarheit beim Definieren von Zielen

selten zu finden, daß Ziele klar und komplett definiert sind - keine Kostenziele, Bemerkungen wie “verbessertes Benutzerservice” oder “bessere Produktverlässlichkeit”, keine Ziele für Erweiterbarkeit oder Produktivität

2.2 Ein Beispiel für ein unklar definiertes Ziel

Das neue Produkt soll folgende Eigenschaften haben: es soll einen angemessenen Kundenservice ermöglichen der die Kunden mit Produkten beliefern kann - das Lager soll auf die Anforderungen des Kundenservice abgestimmt werden (Menge der lagernden Produkte) - dadurch soll mehr Flexibilität und Effizienz in der Verwendung der vorhandenen Ressourcen erreicht werden

Was wollte der Kunde jetzt wirklich?

Beim formulieren der Ziele stellte sich heraus, daß er darauf aus war, 400 Millionen Dollar pro Jahr zu sparen.

2.3 Unklare Ziele

Das Prinzip der unklaren Ziel:

Projekte ohne klare Ziele, werden ihre Ziele auch nicht klar erreichen

(Du kannst nicht ins Schwarze treffen, wenn du nicht weißt wo es ist)

Ziele eines Projekts sind die Ergebnisse die wir wollen. Es ist bei der Definition aber wichtig, daß das eigentliche Ziel nicht mit der Bedeutung, die das Ziel für meinen Lösungsweg hat, vermische. Beispiel: “Steigerung der Performance um das Doppelte” ist ein Ziel. Wie das geschehen soll geht daraus nicht hervor, soll es auch nicht. Dafür sind die Lösungswege da. “Verringerung der Zugriffszeit auf Datensätze, bessere Strukturierung, usw.”

2.4 Wege um die Eigenschaften eines Produktes messen zu können

Um meßbare Eigenschaften zu erhalten, muß man sich überlegen, was die Ziele im praktischen Bereich bedeuten. Manchmal kann das finden von meßbaren Eigenschaften sehr schwer sein, wie zum Beispiel bei diesem Fall:

Wie mißt man Benutzerfreundlichkeit? Man kann ein benutzerfreundliches Programm leicht erkennen, aber wenn man wem anderen sagen soll was er zu tun hat um ein benutzerfreundliches Programm zu schreiben, kann er das meistens nicht.

- wieviel persönliche Erfahrung ist notwendig um in einen Trainingskurs für das Produkt einzusteigen
- wieviel Trainingszeit wird benötigt um ein bestimmtes Produktivitätslevel mit dem Produkt zu erreichen
- wieviel Arbeit kann eine trainierte Person verrichten
- wieviele Fehler werden vom Benutzer in seiner Arbeit erzeugt
- Meinung der Benutzer, wie sehr ihnen das Produkt gefallen hat

Alle diese Punkte können gemessen und klar definiert werden

2.5 Welche Vorteile bringt Meßbarkeit mit sich

Für den Kunden z.B:

- Mehr Gewißheit, das er auch bekommt was er will
- Alle Bewerber um ein Projekt haben die selben Interpretationen, was gefordert wird

Für den Verkäufer z.B:

- Die Möglichkeit, die gewünschten Eigenschaften durch Mißverständnisse zu verfehlen wird reduziert
- Wenn der Auftraggeber die gefragten Eigenschaften ändert, kann man nachweisen das sie sich von den Vorigen unterscheiden und mit einer Änderung des Preises argumentieren.

(Es war doch von Anfang an klar, das wir mit Benutzerfreundlichkeit gemeint haben, das die Arbeitsleistung einer Person um 20 % gesteigert wird)

2.6 Der teuflische Kreislauf der Entwicklung

- Wie die gewünschten Eigenschaften unklar, unkomplett oder falsch sind, wird der Aufbau genauso falsch sein
- Wenn der Aufbau falsch ist, werden die geschätzten Kosten falsch sein
- Wenn die Kosten falsch sind, werden die Leute merken das wie schlecht gemanagt sind

3. Was ist eine Lösung und was nicht?

3.1 Was ist mit einer Lösung gemeint?

Eine Lösung ist eine Sammlung von Ideen, die zumindest einen Teil des Problems positiv behandeln. Eine Lösungsidee ohne positiven Effect auf unsere Anforderungen ist daher auch keine Lösung. Viele Lösungen werden auch negative Seiten haben.

Die Funktion des Produkts wurde von allen Gruppen verstanden (Schaukel auf Baum für Person), die Eigenschaften wurden hingegen nur sehr ungenau festgelegt. Genau das passiert beim Software Engineering. Mit der Zeit wurden zwar die funktionellen Eigenschaften immer intelligenter, aber ohne die Eigenschaften genau zu beschreiben.

Eine Lösung ist immer dann zu beachten, wenn die positiven Beiträge gegenüber den negativen Beiträgen überwiegen. Lösungen sind immer nur solange gültig, solange sie den derzeitigen Anforderungen dienen, und nicht länger. Andere Anf. - andere Lösung.

3.2 Wie unterscheiden sich Lösungen von Zielen?

Zuerst ein nicht technisches Beispiel:

Functional Goals:

- Vater zu sein
- selbständig
- in Paris zu wohnen

Attribute Goals:

- Gesundheit (zumindest 350 nicht bettlägerige Tage im Jahr bis zum Alter von 70)
- Wohlstand (Einnahmen > Ausgaben)
- Weisheit (mehr als zehn neue Bücher pro Jahr lesen)
- Zufriedenheit (mehr als 300 im Jahr an denen man sich positiv und gut fühlt)

Einige Lösungen: (Wege um die Attribute goals zu erreichen)

- einen Gesundheitscheck durchführen lassen
- weniger Geld für Zigaretten und Alkohol ausgeben
- sich entscheiden, sich über Dinge die man nicht beeinflussen kann auch keine Sorgen zu machen

Diese Liste zeigt wie unterschiedlich Funktionen, Eigenschaften und Lösungen sind.

Functional goals listen auf was wir in der Zukunft wollen.

Attribute goals sind variable (in der Zeit) und verhandelbare (kommt darauf an was ich dafür tun muß) meßbare Dimensionen, die die Funktionen besitzen. Sie können variieren wenn sich unsere Anforderungen ändern und unsere Werte und unser Wissen über sie ändert. Die Funktionellen Aspekte bleiben aber ziemlich konstant. Die Lösungen müssen immer von den Zielen getrennt werden, damit sie jederzeit ausgetauscht werden können.

3.3 Definitionen

Functional goals: Zustände die nur einen zukünftigen Zustand haben können: wahr oder falsch

Attribute goals: Zustände die in meßbaren Dimensionen angegeben werden. Sie beziehen sich auf "Qualität" (gute Dinge von denen wir so viel wie möglich haben wollen) und "Ressourcen" (was haben wir zur Verfügung um die Ziele zu erreichen (funkt. & att)

Lösungen: sind Ideen, die die zukünftigen Ziele beinhalten.

Der Software engineering management process konzentriert sich hauptsächlich darauf , die Attribut goals zu erreichen.

3.4 Prinzipien um die richtige Lösung zu finden

Das unheilige Lösung - Prinzip

Lösungen sind niemals heilig - ändern sich die Anforderungen oder ergeben sich Konflikte mit anderen Lösungen sollten sie geändert werden

Das versteckte Lösung - Prinzip

Lösungen sollten nie in einem Ziel beinhaltet sein, außer sie sind wirklich das wichtige Ziel selbst

4. Bewerten von Lösungen

4.1 Die beste Lösung

Lösungen können bewertet und miteinander verglichen werden. Die beste Lösung ist die, welche die meiste Übereinstimmung mit den geforderten Eigenschaften (attribute goals) aufweist.

4.2 Abstimmen der Ziele und Lösungen aufeinander

Zuerst müssen wir feststellen, ob die gewählte Lösung die gewünschten Eigenschaften mit dem gewünschten Niveau erreicht, oder zumindest nicht unter dem Mindestniveau liegt.

Das kann man mit einer einfachen Tabelle.

Ziele	Lösungen
Gesundheit	nicht rauchen, nicht trinken, früh ins Bett
Wohlstand	nicht rauchen, nicht trinken, spart 10 % des Einkommens
Weisheit	keine Lösung
Zufriedenheit	humorvolle Bücher lesen, nett zu anderen Leuten sein, früh aufstehen

Hier sehen wir, das mehr als eine Lösung pro Eigenschaft angewandt wird und einige Lösungen können mehr als eine Eigenschaft abdecken. Weisheit hingegen ist durch keine Lösung abgedeckt, das ist ein Zeichen, das Weisheit mehr Aufmerksamkeit braucht, wenn wir unsere Ziele erreichen wollen.

Aber diese Analyse sagt uns viele Dinge nicht, die wir noch gerne wissen würden. Welche Nebeneffekte haben die Lösungen? Ein Raucher und Trinker wird der Lösung einen negativen Nebeneffekt auf seine Zufriedenheit zusprechen. Die Lösung nett zu anderen Leuten zu sein, könnte beinhalten, anderen Leuten einen Drink anzubieten, und dann aus Höflichkeit mitzutrinken. Die Tabelle gibt uns einige Ideen, wie sich die Lösungen auf die Ziele auswirken werden und ist einer mündlichen Beschreibung der Lösungen immer vorzuziehen.

4.3 Die Lösungen vergleichen

Es ist fast unmöglich, ein Set von Zielen anzugeben ohne

- eine Liste mit viele Zielen, statt ein oder zwei zu bekommen
- herauszufinden, das jedes Ziel in irgend einer Weise in Konflikt mit einem anderen steht
- herauszufinden, das einige Ziele wichtiger sind als andere, und das macht das aussortieren der Konflikte zwischen den Zielen noch schwieriger

A, B, C A beeinflusst B, C negativ - A ist aber wichtiger als B und C

Beim vergleichen der Lösungen soll jetzt die best passende Lösung, für die vielen geforderten Eigenschaften gefunden werden.

Normalerweise haben wir sehr viele Informationen über jede alternativer Lösung. Zum Beispiel beim Kauf neuer Hardware. (Höhere Taktrate, Prozessor, Speicher,...)

Tabelle

	Lösung 1	Lösung 2	Lösung 3	Summe
Attribut 1	100	40	0	140
Attribut 2	-40	0	80	40
Attribut 3	30	5	45	80
Lösungsvergleich	90	45	125	

Lösung 3 die beste, weil sie die 3 Attribute zusammen am besten erfüllt

Summe der Attribute zeigt, wie sehr sich die Lösungen mit jedem Attribut beschäftigen
 Prozentzahlen, wie gut das Attribut erfüllt wird

Die Tatsache, das alle Daten übersichtlich aufgelistet sind, macht das finden der richtigen Entscheidung einfacher

5. Das Risiko abschätzen

5.1 Worin besteht das Risiko?

Die Umsetzung von Lösungen, die auf Plänen aufbauen, die auf Schätzungen, Annäherungen und Unklarheiten aufbauen, sind mit Risiko verbunden. Aber auch in der Wirtschaft gilt, je höher das Risiko, desto größer der Gewinn. Es ist daher sinnlos zu versuchen, das Risiko zu

eliminieren, oder zu versuchen es zu verkleinern, es ist wichtig die richtigen Risiken einzugehen.

5.2 Für das Risiko verantwortlich sein

Man kann das Ergebnis von Aktionen schon relativ früh abschätzen, aufgrund von Beobachtungen. Man muß dabei aber bereit sein, ein bestimmtes Risiko auf sich zu nehmen. Annahmen könnten falsch sein oder irreführend. Wenn wir den Fehler machen, auf 100 % richtigen Abschätzungen zu bestehen, wird das auch ewig dauern. Wir sollten uns vielmehr darauf konzentrieren, das Risiko zu kennen und zu kontrollieren. Zu wissen, daß ein Risiko vorhanden ist, und auch das Level dieses Risikos zu kennen ist ein Zeichen von Kompetenz. Manche Fachleute glauben irrtümlich, daß das aufzeigen von Risiken ein Zeichen von Inkompetenz ist - weil es zeigt das man keine vollkommene Kontrolle hat. Es ist aber wichtig das Risiko zu kennen und anderen mitzuteilen, auf was sie sich einlassen.

Das Risiko teilen - Prinzip

Ein wirklicher Fachmann kennt das Risiko, den Grad des Risikos, die Ursachen, die Handlungen die notwendig sind um das Risiko zu kontrollieren und teilt sein Wissen seinen Kollegen und Klienten mit.

5.3 Überschreitungen erwarten

Das gilt vor allem im Bereich der Kosten. Von einem professioneller SW-engineer wird erwartet, das er zwischen vorhersehbaren Kosten (durch Erfahrung) und zwischen unsicheren Kostenfaktoren (neue Techniken, keine Erfahrung über Kosten) unterscheidet. Wie kann man jetzt als Manager die schlimmsten Effekte eines unbekanntes Systems verhindern.

Das Versprechen - Prinzip

Gib niemals ein Versprechen das du nicht halten kannst, auch nicht unter Druck

Das niedergeschriebenen Versprechen - Prinzip

Wenn du ein Versprechen gibst, dann von dir aus, und in niedergeschriebener Form

Das stillschweigende Versprechen - Prinzip

Wenn du glaubst, das jemand anderer (Boß oder Kunde) annimmt das du ein Versprechen gemacht hast, nimm dir die Zeit um es abzustreiten, und die Versprechen zu wiederholen, die du wirklich gemacht hast, und zwar schriftlich

Das Abweichungs - Prinzip

Wenn du eine Abweichung feststellst, mache eine Liste der möglichen Ursachen und der möglichen Aktionen um diese Risiken zu kontrollieren

Das beweis es schriftlich - Prinzip

Hänge das folgende Schild bei deinem Schreibtisch auf: Wenn du es nicht schriftlich von mir hast, habe ich es nicht versprochen

5.4 Den Grad der Unsicherheit bestimmen

Management beinhaltet immer das Treffen von Entscheidungen, mit unterschiedlichen Graden des Risikos das zu Fehlentscheidungen führen kann. Das Risiko muß aber klar definiert werden. Unbrauchbar ist die Aussage: Muß so bald wie möglich fertig sein. Eine Verbesserung wäre es zu sagen: muß am 24. Dezember um 17:00 fertig sein. Aber es wäre noch sinnvoller, eventuelle Unregelmäßigkeiten festzuhalten. Wie z.B.: Es wird geplant das Projekt bis zum 24. Dezember fertigzustellen, dieser Termin ist der früheste vorstellbare Termin. Es ist eine große Wahrscheinlichkeit gegeben, das es durch Krankheit des Personals zu einer einwöchigen Verspätung kommen wird. Weiters ist auch eine Verzögerung um ein Monat denkbar, wenn der Hardwarelieferant es versäumt wie ausgemacht zu liefern. Zusammengefaßt: 24. Dez. (+7, Personalkrankheit) und (+31 Lieferschwierigkeiten des Hardwarelieferanten)

Weiters ist zu beachten, daß das Risiko steigt, wenn es sich um eine Neuentwicklung handelt. statt 20,000 Transactionen 40,000 durchführen will. Die höchste bekannte Zahl von Transactionen auf einem ähnlichen System sind aber 30,000. Das Projekt-Team muß etwas neues, innovatives Umsetzen.

Das Prinzip der Risikoaufdeckung

Der Grad des Risikos und seine Ursachen, dürfen nie vor den Entscheidungstreffern versteckt werden.

Das Fragen - Prinzip

Wenn du nicht nach Risikoinformationen fragst, fragst du nach Problemen.

6. Evolutionäre Lieferung

Die meisten Bücher und heutigen SW engineering models sind auf dem "Wasserfallmodell" für die Lieferung aufgebaut. Das kann verschiedene Formen annehmen, aber charakteristisch dafür ist:

- das die Planung des ganzen Projekts auf einen Liefertermin ausgelegt ist. Wenn die Lieferung in einzelne Phasen aufgeteilt ist, dann sind sie sehr umfangreich. Es gibt kein Konzept um unzufrieden abgeschlossene Phasen noch einmal aufzuarbeiten. (Macht es so gut es geht fertig, und aus)
- alle Analysen und das ganze Design werden bis ins Detail vorbereitet, bevor mit der Programmierung und dem Testen angefangen wird

Die Methode der evolutionären Lieferung basiert auf den folgenden einfachen Prinzipien:

- Liefere etwas an einen wirklichen End-user
-
- richte das Design nach den Erfahrungen aus, die du beobachtet hast

Das einfachste SW-Model von evo Lieferung, ist ein Personal Computer Benutzer, der sich seine eigene Applikation zusammenstellt. Der Programmierer wird zu einem Anwender, dem Zielsetzer, Analytiker, Designer und Tester. Es ist eine ständige Verbesserung von der ersten Phase an, während das System entsteht. Das System das entsteht entwickelt sich ständig weiter. Der Benutzer kann das Design ändern, Programmdetails was den Code betrifft ändern und auch die Ziele. Evolutionäre Lieferung besteht aus einer Sammlung von vielen Konzepten. Die grundlegendsten lauten wie folgt:

- Multi-objectives
- frühe, ständige Verbesserung
- komplette Analysen, Design, Entwicklung und Test bei jeder Phase
- User orientiert
- System orientiert, nicht hauptsächlich auf Algorithmen konzentrieren
- open-ended basic systems architecture

6.1 Planen für multiple objectives

Die übliche Sw-Planung ist überwiegend "Funktions" orientiert. Die Planung geschieht auf Grund von funktionellen Ansprüchen (Was die SW können wird) statt sich vielmehr um das **Wie gut und zu welchem Preis** wird es das können zu kümmern. Heutiges SW-engineering beschäftigt sich viel zu wenig mit der Kontrolle der kritischen Qualitäts- und Resource Eigenschaften eines Systems, und deswegen verliert es die Kontrolle über diese Eigenschaften. Ein einfaches Beispiel ist das Fehlen von Wissen unter Software engineers, wie man kritische Eigenschaften wie "Benutzerfreundlichkeit" oder "Wartbarkeit" definiert. Diese Dinge sind aber so wichtig für den Erfolg eines heutigen SW-Projects, das die Ignoranz davon einem Projekt großen Schaden anrichten kann.

Evolutionäre Lieferung ist auf der Weiterentwicklung entlang von klar und meßbaren Objectives aufgebaut. Das Set von Objectives muß alle funktionellen, qualitativen und resource objectives beinhalten, die für das langzeitige und kurzzeitige Überleben des Systems unter Entwicklung notwendig sind. Projekte die nicht so aufgebaut sind, sind auch nicht nach den Anforderungen des Anwenders aufgebaut.

6.2 Frühe und ständige Verbesserung

In den meisten SW-engineering Projekten, ist der Zeitplan so festgelegt, das frühestens nach einem Jahr ein erstes praktisches und brauchbares Ergebnis vorhanden ist. Es gibt zwar einige Gründe für das Fehlen einer frühen Konfrontation mit dem echten Anwender, aber die Ausreden davor sind unzureichend. Das Management eines solchen Projekts würde natürlich gerne die ersten Ergebnisse so früh wie möglich haben. Aber das selbe Management akzeptiert auch die allgemeine Einstellung, das es einem langen Kreislauf bedarf, bevor etwas nützliches geliefert werden kann.

Das Prinzip der evolutionären Lieferung basiert aber darauf, das das Entstehen des Systems in viele kleine Phasen zerlegt wird, deren Ergebnis von einem Anwender getestet wird, und dessen Urteil in die Weiterentwicklung mit einbezogen wird. Die Auswahl der Reihenfolge der Phasen kann so getroffen werden. "Welche Phase benötigt den wenigsten Aufwand, bewirkt aber trotzdem etwas in Richtung unserer Ziele."

6.3 Komplette Analysen, Design, Entwicklung und Test nach jeder Phase

Eine der großen Zeitverschwendungen in SW-Projekten sind detaillierte Bedürfnisanalysen, gefolgt von detailliertem Design. Wie können solche Aufgaben nur für sehr kleine Projekte bestimmen. Es sind so viele Unbekannte, zu viele dynamische Veränderungen, und ein zu komplexes Set von Weiterentwicklungen in den Systemen die wir entwickeln. Wir müssen etwas bescheidener sein.

Wir müssen meßbare Objekte festlegen, soweit wird eine begründete Meinung über die Zukunft haben. Wir müssen bereit sein, die Objekte zu verändern, sobald wir aus den Erfahrungen durch die Lieferung der einzelnen Phasen dazu gedrängt werden. Wir müssen meßbare Objectives für jede kleine Lieferungsphase definieren. Ist einfach nicht möglich ein Set von multiple quality, ressource und functional objectives zu definieren, und sicher zu sein, das alle wie geplant erreicht werden. Man muß darauf vorbereitet sein, Kompromisse einzugehen.. Wir müssen die technische Lösung designen, entwickeln, testen, liefern und Feedback bekommen. Dieses Feedback muß dafür verwendet werden, um das vorhandene Design, falls notwendig, zu verändern, und um sowohl Kurz- als auch Langzeitziele zu ändern. Es ist sinnlos so viel Geld auszugeben, oder vielmehr zu vergeuden, um über Bedürfnisse und technischen Eigenschaften zu spekulieren, die viel leichter bestimmt werden können, wenn es während der Implementierung eines funktionierenden Systems geschieht. Evolutionäre Lieferung soll uns frühe Warnsignale liefern. Die sind zwar unangenehm, verhindern aber, das Unzulänglichkeiten in unserem Produkt nicht zu groß werden

6.4 User-Orientiertheit

SW Projekte werden nicht so sehr wegen der erfolgreichen Einstellung auf den Markt oder den Benutzer für gut befunden. Die Orientierung liegt viel mehr Richtung der Maschine, den Algorithmen, oder den Deadlines - aber viel zu selten Richtung User. Viele SW-Entwickler sehen ihr Produkt nicht einmal im Einsatz mit richtigen Anwendern, genausowenig wie Anwender die Gesichter von Designern oder Programmierern sehen. Selbst wenn der Entwickler es gewollt hätte, ein Produkt zu entwickeln das jeden Benutzer glücklich macht, ist es meistens schon zu spät oder zu unpraktisch es zu tun, nachdem zuviel Zeit vergangen ist, um es herauszufinden.

Mit evolutionärer Lieferung hat sich die Situation verändert. Der Entwickler ist mit dem Anhören der Benutzerreaktionen beauftragt, sehr früh und oft. Der Benutzer kann also eine wichtige Rolle im Entwicklungsprozeß übernehmen. Wertvolle Benutzerideen können die Planern auf eine Reihe von neuen Ideen bringen, die ursprünglich nicht geplant waren. Wenn also bessere Ideen vorhanden sind, müssen wir sie so früh wie möglich einbauen. Jeder Systementwickler sollte begreifen, das er Feedback braucht

6.5 Systemorientiert, nicht hauptsächlich auf Algorithmen konzentriert

Viele SW-engineering Methoden haben eine gemeinsame Schwäche. Sie orientieren sich rein auf aktuelle Programmiersprachen. Sie kümmern sich herzlich wenig um den Aufbau der Daten, und noch viel weniger um offensichtliche Anliegen wie Dokumentation, Training, Marketing oder Motivation.

Evolutionäre Lieferung ist eine Methode die nicht nur auf SW limitiert ist. Sie ist für jeden kreativen Prozeß geeignet. Man darf nicht vergessen, dem ganzen Aufbau der Systemarchitektur zu betrachten, in dem die Software nur ein Teil ist.

6.6 Open-ended basic systems architecture

Unter den prinzipiellen Attributes eines jeden Systems sind die, die das Überleben und Erreichen der Ziele unter Konditionen erlauben, die sich mit der Zeit ändern. Ein guter SW-engineer sollte fortwährend und detailliert die vorhandenen Design Technologien studieren, die zu Systemen führen die anpassungsfähiger sind als andere. Es gibt eine Reihe von meßbaren, technischen Eigenschaften, wie zum Beispiel Wartbarkeit, Transportierbarkeit und Erweiterbarkeit.

Was die evolutinäre Lieferungsmethode betrifft, sind offene Architekturen absolut notwendig. Eine offene Architektur muß es erlauben, auf Änderungen reagieren zu können..

6.7 Die Gemeinsamkeiten mit Auto fahren und Schach

auf plötzliche Veränderungen eingehen - Wenn du ein Auto fährst, mußt du vorausdenken und planen. Du kannst aber nie absolut sicher sein, das du den gewählten Weg auch fahren kannst, weil sich etwas Unvorhergesehenes ergibt.

Trip von London nach Rom - Vergeudetet Zeit - minimum an Planung - Route festlegen - Straßenkarte einstecken und Extrazeit einplanen für event. Zwischenfälle.

Schachspielen - hat das Ziel zu Gewinnen - bestimmte Strategie und Taktiken - aber bei jedem Zug des Gegners mußt du sie neu überdenken