2. Der Software Prozess und seine Probleme

Unter Software Prozess verstehen wir im Folgenden:

- Den Software Lebenszyklus
- Die Werkzeuge
- Die Beteiligten

Der Software Prozess ist bei jeder Firma unterschiedlich. Aber trotzdem gibt es Gemeinsamkeiten aller Software Prozesse. Auf die kommst es uns hier an!

In jeder Phase müssen bestimmte Aktivitäten durchgeführt werden. Daneben gibt es Aktivitäten, die in jeder der Phasen durchgeführt werden müssen:

- Dokumentation erstellen
- Testen, ob das Ergebnis mit den Eingangsanforderungen konform ist

Worum geht es in diesem Kapitel?
Wir möchten aus der Vogelperspektive einen Überblick über den Software Prozess (Entwicklung und Wartung) gewinnen. Die einzelnen Begriffe und Konzepte, die Phasen und Techniken werden wir in den folgenden Kapiteln noch im Detail und vertieft kennen lernen. Wir werden auch verschiedene Konzepte beispielhaft, wenn auch nicht immer sinnvoll, in Java implementieren.
Ihre Aufgabe im Zusammenhang mit der Java Programmierung wird es sein, zum einen eigene Programme zu entwickeln, zum andern Programme zu analysieren und abzuändern.

2.1. Kunden, Entwickler und Benutzer

Zuerst einige Definitionen:

Der Kunde:
Kann eine individuelle Person sein oder eine ganze Organisation. Der Kunde benötigt unsere Software, um damit sein Problem zu lösen.

Der Entwickler:
Der Entwickler produziert unser gewünschtes Software System. Unter Umständen ist der Entwickler nur für eine bestimmte Phase der Entwicklung zuständig, oder er ist nur für ein bestimmtes Teilsystem, aber alle Phasen zuständig.

Software Entwicklung:
Der Begriff Software Entwicklung umfasst alle Aspekte zur Herstellung unseres (Software) Produktes.

Falls der Kunde und der Entwickler zur selben Organisation gehören, dann handelt es sich um ein *internes* Software Projekt; im andern Fall muss zwischen dem Entwickler und dem Benutzer ein Software *Kontrakt* bestehen, der die Entwicklung und die Wartung umfasst.

2.2. Requirement Phase

Software Entwicklung ist teuer! Normalerweise beginnt der Entwicklungsprozess damit, dass der Kunde den Entwickler bittet bestimmte Software zu erstellen, in der Hoffnung, dass er (der Kunde) damit einen wirtschaftlichen Vorteil erringen, vergrössern oder zumindest erhalten kann.

In einem ersten Meeting wird der Kunde dem Entwickler erklären, worum es geht. Aus der Sicht des Entwicklers ist diese Beschreibung in der Regel recht ungenau und nur bedingt brauchbar. Der Entwickler muss also zusätzliche Informationen sammeln und erfragen. Dies umfasst neben den funktionalen und evtl den ergonomischen Aspekten auch den Budgetrahmen und zeitliche Vorstellungen. Diese Vorabklärungen werden in der Regel als „*Concept Exploration*“ bezeichnet, oder eben Vorabklärungen. Die Anforderungen an das neue System werden nun in Folgebesprechungen schrittweise vertieft und systematisiert. Die technische Machbarkeit wird abgeklärt und der wirtschaftliche Rahmen wird überprüft.

Das scheint alles klar und fast banal zu sein. Der Haken ist : diese Phase wird in der Regel sehr unsorgfältig durch geführt. Der Entwickler hat das Gefühl, er wisse schon, was der Kunde sich so vorstellt! Der Kunde gewinnt den Eindruck, der Entwickler verstehe das Kundengeschäft: die Sprache der beiden scheint zu überlappen; aber eben, es scheint nur so!

Einer der Hauptgründe, warum so allerlei schief geht, liegt an der Tatsache, dass Software sehr viele Freiheitsgrade hat und sehr komplex sein kann.

Mit Hilfe von *Rapid Prototyping* kann man versuchen, dem Benutzer möglichst schnell etwas zu zeigen. Er wird fast sicher damit nicht sehr glücklich sein; aber die Ideen werden dann beiden Seiten (Entwickler und Kunden) klarer. Eine Gefahr des Rapid Prototyping ist, dass der Benutzer schnell etwas sieht, was sehr ähnlich aussieht, wie das Endergebnis. Er versteht nun häufig überhaupt nicht, warum den die restliche Programmierung noch so lange dauern soll!

2.2.1. Requirement Phase Tests

Im Idealfall verfügt das Entwicklungsteam über einige Spezialisten, die sich ausschliesslich um das Testen und Verifizieren (Verifizieren: entspricht das Ergebnis den Vorgaben?) kümmern kann, einer Software Qualitäts Gruppe. Die Mitglieder dieser Gruppe sind in der Regel, die am höchsten qualifizierten Mitarbeiter.... zumindest sollten sie das sein. Die Realität sieht anders aus: Neueintretende werden zur Software Qualitäts Gruppe eingeteilt, damit sie sich möglichst schnell mit allen Einzelheiten der Software vertraut machen können.

Im Idealfall bespricht der Entwickler mit dem Kunden nach Abschluss der Anforderungsphase die Requirements und lässt sich vom Kunden bestätigen, dass dies seine Anforderungen sind. In der Realität bringt dies nicht sehr viel, weil sich die Anforderungen im Laufe der Entwicklung vermutlich ändern werden. Speziell im Falle einer Firmenübernahme muss damit gerechnet werden, dass von zentraler Stelle neue Anforderungen an das Entwickler - Team heran getragen werden, sofern das Projekt überhaupt weiter geführt wird!

2.3. Spezifikations Phase

Nachdem der Kunde in der Anforderungsphase seine Bedürfnisse eingebracht hat, wird die *Spezifikation* erstellt.
Im Gegensatz zur eher informellen und zum Teil auch noch groben Anforderungsphase, muss in der Spezifikations - Phase das zu erstellende System möglichst genau umschrieben werden:

- Seine Funktionalität
- Ergonomie Anforderungen
- Technologie Anforderungen
- Zeit und Kosten

Die Spezifikation

- Eingaben
- Ausgaben
- Evtl spezielle Algorithmen
- Ausnahmefälle

Die Spezifikation bildet den eigentlichen Kontrakt zwischen dem Auftraggeber und dem Auftragnehmer. Aus diesem Grunde muss in der Spezifikation eine klare und eindeutige Sprache verwendet werden. Speziell im Falle eines Abbruchs der Entwicklung kann es zu einer gerichtlichen Auseinandersetzung kommen. In diesem Falle kann die Spezifikation DAS entscheidende Dokument sein!

Während der Spezifikations - Phase können sehr unterschiedliche Probleme auftauchen:

- Die Spezifikation kann zu ehrgeizig sein
- Die Technologie kann noch nicht bereit sein (zu langsam, zu ungenau, ...)
- Bestimmte Formulierungen sind nicht eindeutig
- Die Spezifikation ist unvollständig
- Fehlerfälle sind nicht beschrieben
- Ausnahmefälle sind unvollständig
- Anforderungen widersprechen sich

2.3.1. Testen in der Spezifikationsphase

Gemäss den bereits besprochenen Statistiken über Fehler und deren Konsequenzen, treten Fehler sehr häufig in der Spezifikations - Phase auf. Es lohnt sich also im Sinne einer Qualitätssicherung einen Review durch zu führen, im Sinne eines Spezifikations- Tests

Mögliche Ansatzpunkte für einen Spezifikations- Test

- Gibt es widersprüchliche Anforderungen
- Sind die Anforderungen wirklich vom Kunden (*traceability)*
- Wurde Rapid Prototyping eingesetzt: sind alle Änderungswünsche besprochen, zurück gewiesen oder eingeflossen

Durchführung eines Reviews:

- Besteht die Möglichkeit, den Kunden und die Entwickler zu einem gemeinsamen Workshop zusammen zu bringen, in dem alle Anforderungen nochmals durch diskutiert werden
- Der Moderator des Workshops muss entsprechend geschult sein und als Moderator, nicht als Kunde oder als Entwickler teilnehmen.

2.4. Planungs Phase

In der Regel verlangt der Kunde vor der Realisierung des Projektes einen detaillierten Plan. Aus dem Plan müssen die Meilensteine ersichtlich sein, sowie der Aufwand und die benötigten Resourcen:

- Zeit
- Geld
- Personen

Je nach Vertrag wird der Kunde einen fixen betrag für das Produkt bezahlen. Dann muss die Planung besonders sorgfältig durch geführt werden, da sonst die eine oder die andere Partie sich über den Tisch gezogen fühlt!

Die Projekt Planung bei Software Projekten unterscheidet sich wenig bis nicht von der Projekt Planung in anderen Bereichen. Eine Planung benötigt jedoch Input:

- Erfahrungswerte für die Realisierung bestimmter Funktionen (aus vergangenen Projekten oder der Literatur)
- Standard Projektpläne für die Abwicklung von bestimmten Projekten (Einführung einer bestimmten Software, Umstellungsaufwand auf ein bestimmtes Betriebssystem oder ein bestimmtes neues Release)
- Persönlichkeitsprofile, da diese die Produktivität der einzelnen Mitarbeiter für bestimmte Aufgaben aufzeigen

Der Projektplan kann seriös erst erstellt werden, nachdem die Spezifikation bekannt ist. Der Projektplan muss im Speziellen auch eine Beschreibung enthalten, was abgeliefert wird, alle Meilensteine (wann, was) und das Budget.

Unter Umständen kann der Projektplan auch Informationen über einzusetzende Werkzeuge (CASE Tools, Programmier-Tools, Debugging Tools, Modellierungs-Tools) enthalten.

2.4.1. Testen in der Planungsphase

Da der Projektplan über Resourcen spricht, ist es zentral zu prüfen, ob diese auch und ausreichend zur richtigen Zeit zur Verfügung stehen (Ferien, Feiertage, Krankheit, Militär, soweit planbar).

Das Budget kann in der Regel auf zwei Arten bestimmt werden:

- Top down
  indem zuerst eine grobe Aufteilung auf die wichtigsten Projektteile oder Phasen vorgenommen wird
- Bottom up
  indem die einzelnen Aktivitäten einzeln geschätzt werden und dann kumuliert wird

Beide Methoden sollten zu einem vergleichbaren Ergebnis führen!

2.5. Design Phase

Die ***Spezifikation*** des Produktes beschreibt das ***WAS,***.

Das *WIE* wird in der *Design-Phase* festgelegt.
Als Ausgangspunkt dient die Spezifikation. Daraus leitet das Design-Team die Struktur des zu schaffenden Systems her. Spezielle Algorithmen und allfällige speziell einzusetzende käufliche Module werden fixiert und ins Design eingebunden.

Das System wird in *Module* aufgeteilt, wobei jeder Modul so unabhängig von den andern Modulen sein sollte wie nur irgend wie möglich! (Parnas Prinzip)
Pro Modul wird festgehalten, was dieser Modul für eine Aufgabe hat, welche Eingaben er erwartet und was er produziert oder verändert.

Das Design-Team muss auch festhalten, warum bestimmte Anforderungen zu einem Modul zusammen gefasst wurden. Dadurch ist es jederzeit möglich zu einem späteren Zeitpunkt auf die Design-Entscheide zurück zu kommen.

Die Design-Entscheide sind für den ganzen Lebenslauf der Software von entscheidender Bedeutung. In der Wartungsphase, in der die ursprünglichen Entwickler vielleicht bereits nichts mehr mit dem Produkt zu tun haben, kann der Wartungs-Engineer jederzeit nachlesen, warum bestimmte Algorithmen eingesetzt wurden.

In der Praxis wird dies kaum praktiziert. Im MIL Bereich ist dies jedoch in der Regel Teil des Kontraktes!

Das Hauptergebnis der Design Phase ist der Design! Dieser besteht aus zwei Teilen:
- Dem Architektur Design
  in ihm wird die Zerlegung des Systems in logische Einheiten, Module bezw. Objekte beschrieben
- Dem Detail- Design
  in ihm werden die einzelnen Module im Detail beschrieben

Die Beschreibung der Module dient den Programmierern als Vorgabe bei der Implementierung. Wir werden später Beispiele kennen lernen und die Zerlegung eines Systems in Module bezw. den Objekt - Design explizit üben.

2.5.1. Design Phase Tests

Tests in der Design-Phase sind recht kritisch! Analog zum Review in der Spezifikations Phase, kennt man auch in der Design-Phase diverse Reviews und sogenannte Walkthroughts, also ein konzentriertes Besprechen des gesamten Designs.

Zusätzlich muss die „Traceability“ beachtet werden:
- Sind alle Anforderungen in das Design eingeflossen?
- Sind die Module möglichst entkoppelt und enthalten nur die Informationen, die unbedingt benötigt werden?
- Sind die Schnittstellen eindeutig spezifiziert?
- Sind Ausnahmen abgedeckt?
- Entspricht das Design der Spezifikation?

Beim Design Review ist der Kunde in der Regel nicht anwesend, da die Terminologie des Designs bezw. des Design-Teams in der Regel nicht oder nur bedingt versteht! Wir werden später einen Design-Review durch führen!

2.6. Implementations Phase

In der Implementations-Phase werden die Module aus der Design-Phase mit Hilfe einer (passenden) Programmier-Sprache „implementiert“. Wir werden uns selbstverständlich damit im Detail beschäftigen. Die Dokumentation der Implementations-Phase ist sicher primär der Source-Code, hoffentlich mit Kommentaren versehen. Es gibt für fast jede Programmiersprache:

- Formattierer, die den Source-Code in eine Standard-Form bringen
- Text-Extrahierer, die Kommentare aus dem Source-Code extrahieren und eine Art Dokumentation erstellen

Zusätzlich sollten alle Testfälle dokumentiert werden, damit bei Änderungen oder im Fehlerfall beim Kunden, mit Hilfe des definierten Testsatzes das System jederzeit wieder überprüft werden kann!
Die Tests müssen Test-Eingaben und Test-Ausgaben enthalten und allfällige Hinweise, die erklären, warum der Test ausgewählt wurde.

2.6.1. Testen in der Implementations-Phase

Das Testen sollte durch unterschiedliche Personen geschehen. Als Erstes durch die Personen, die die Module implementiert haben („desk checking“). Dann muss die Qualitäts-Sicherungs-Gruppe mit Hilfe systematischer Tests, die wir noch besprechen werden, das System testen. Insbesondere müssen Anwendungsfälle des Benutzers durch gespielt werden.

Häufig wird zusätzlich ein Code-Review durch geführt. Allerdings fast nur dann, wenn genügend Zeit vorhanden ist bezw. falls die Software „Mission Critical“ ist dh. Wenn vom Funktionieren oder Nichtfunktionieren das Leben anderer Menschen abhängt!

2.7. Integrations Phase

In einem nächsten Schritt müssen diese Module zu einem Gesamtsystem zusammen gefügt werden! Wie und in welcher Reihenfolge die Module zusammen gefügt werden, hat einen recht grossen Einfluss auf die Qualität der Software.

Falls die Integration bottom up geschieht und sich ein Design-Fehler zeigt, dann führt dies zu einen recht grossen Re-Integrations-Aufwand.
Bei einer top down Integration, bei dem unter Umständen die einzelnen Module noch nicht fertig implementiert sind, aber das Zusammenspiel bereits getestet wird, kann ein Design-Fehler leichter korrigiert werden. Diese Art der Implementation wird explizit durch bestimmte Programmiersprachen unterstützt (Ada als Beispiel).

2.7.1. Integrations Tests

Das Software-Testen ist eigentlich eine gut ausgebaute und erforschte Technik. Allerdings verlangt sie genügend Zeit und ein Committment des Managements. In der Regel muss Software sofort zum Kunden:

- Damit die Firma Wartungs-Gebühren einziehen kann
- Damit die grossen Probleme im bestehenden Release endlich behoben werden
- Damit der Druck der Kunden endlich wieder erträglich wird

In der Regel ist der Kundendruck so gross, dass die Software kaum fertig und kaum getestet zum Kunden ausgeliefert wird („Bananen-Software“ : die Software wird beim Kunden reif).
Das Software-Testen ist eigentlich eine gut ausgebaute und erforschte Technik. Allerdings verlangt sie genügend Zeit und ein Committment des Managements. In der Regel muss Software sofort zum Kunden:

- Damit die Firma Wartungs-Gebühren einziehen kann
- Damit die grossen Probleme im bestehenden Release endlich behoben werden
- Damit der Druck der Kunden endlich wieder erträglich wird

In der Regel ist der Kundendruck so gross, dass die Software kaum fertig und kaum getestet zum Kunden ausgeliefert wird („Bananen-Software“ : die Software wird beim Kunden reif).
Bei neuer Software kann der Kunde einen Abnahme-Test verlangen. In diesem Falle wird der Kunde seine Anforderungen gegenüber dem abgelieferten Software-Produkt verifizieren.

Als erster erhält der Kunde oft eine Alfa Version. Diese ist noch recht unvollständig, kann aber bestimmte Aspekte der Software bereits voll implementieren. Alfa Software ist KEIN Prototyp. Aus der Alfa Version entsteht die Beta Version. Beim Beta Testen wird dem Kunden eine noch nicht voll getestete Version zur Verfügung gestellt. Der Benutzer testet die Software an seinen typischen Geschäftsfällen, sicher weniger Entwickler-blind als der Programmierer. Beta Testen ist besonders wichtig, falls die Software an eine breite Käuferschaft verkauft werden soll.

2.8. Wartungs-Phase

Nachdem wir nun glücklich und zufrieden die Software beim Kunden „abgeladen“ haben, beginnt die Wartungs-Phase. Häufig bereits zum Zeitpunkt der Installation beim Kunden, wird der Entwickler die eine oder andere Erweiterung vornehmen oder ein Modul, von dem er weiss, dass es eigentlich nicht so funktioniert wie die Spezifikation vorgab, weiter entwickeln.

Das gleiche Problem tritt auch bei Webpages auf, die ja in der Regel von Studenten als Nebenverdienst erstellt werden: Webpages sind in der Regel nicht fertig, obschon sie auf dem Internet sind. Es kann auch sein, dass bestimmte Effekte (layers, dynamisches HTML) beim einen Browser bestens funktioniert, und dies ist der Vorzugs-Browser des Kunden; beim andern Browser ist der Effekt jedoch nicht korrekt implementiert (layers , Stylesheets und DHTML unterscheiden sich beträchtlich bei Netscape und Microsoft).

Damit der Entwickler auch nach seinen Ferien, oder einem anderen Projekt, seinen eigenen Code noch versteht, muss nicht nur der Programm-Code dokumentiert und kommentiert werden, sondern auch Änderungen jeweils mit einem Datum versehen werden und dokumentiert sein!

Im Falle von Middleware, Software im System oder Netzwerk- Bereich, ist es sehr wesentlich zu dokumentieren, unter welchen Releases die Software entwickelt und getestet wurde. Java war in dieser Beziehung am Anfang besonders schlimm: Java Programmierer prägten den Satz „am Morgen entwickeln und am Nachmittag an der selben Maschine zeigen“, da die Änderungen in der Virtual Machine und den classes.zip laufend geschahen und der Benutzer kaum darüber informiert wurde. Das Problem war aber immer durch Regenerierung des Bytecodes zu beheben.

2.8.1. Testen in der Wartungs-Phase

Das Testen in der Wartungs-Phase bedingt ZWINGEND, dass bestehende Testfälle nachvollzogen werden. Deswegen müssen Testfälle im Detail dokumentiert sein!

In der Regel entstehen Probleme in der Wartungsphase aus folgenden Gründen:

- Der Entwickler denkt, er kennt noch jedes Detail und baut eine Änderung ein; das aktuelle System verhält sich aber anders. Die Änderung produziert einen Fehler oder sogar einen Absturz
- Der Entwickler ist nicht mehr in der Firma. Sein Nachfolger kennt das System zuwenig und ändert unbewusst die Schnittstelle des fehlerhaften Moduls und auch einen Aufruf, der den Fehler verursacht hat. Das Modul wird aber auch noch von andern Modulen aufgerufen. Der neue Entwickler hatte schlicht die Übersicht nicht und konnte die Konsequenzen seiner Änderung nicht abschätzen

Ein reales Beispiel:
In einem Qualitätssicherungs-System (MIL Bereich) werden alle Datenrecords mit einen „1“ Flag in das File CAAF geschrieben. Falls das Flag nicht gleich „1“ ist, wird nichts geschrieben. Der Entwickler hat schon längere Zeit nicht mehr an dem System gearbeitet und vertauscht die Bedeutung des Flags. Alle Daten werden in kürzester Zeit vernichtet.

2.9. Retirement / Ablösung

Ablösungen können unterschiedliche Effekte auf bestehende Software haben

- Die Konzepte der bestehenden Software werden übernommen
- Die Konzepte sind überholt und werden durch neue ersetzt

In beiden Fällen ist es sehr wesentlich, dass das bestehende System so gut dokumentiert ist, dass im ersten Fall auf die Dokumentation zurück gegriffen werden kann. Im zweiten Fall ist die Dokumentation eigentlich noch wichtiger, da dadurch Designfehler vermieden werden können.

2.10. Probleme mit dem Software-Prozess

Über die letzten 30 Jahre wurde die Hardware immer billiger. Auf der andern Seite wuchsen die Anforderungen an die Hardware beträchtlich. Das hat dazu geführt, dass ein PC, obschon immer leistungsfähiger, immer etwa gleich teuer bleibt (seit mehreren Jahren).

Brooks hat in einem immer wieder zitierten Artikel (Sie müssen den Artikel lesen) „No Silver Bullet“ untersucht, was Gründe dafür sein könnten, dass sich die Software nicht genau so exponentiell entwickelt hat.
„... building software will always be hard. There is inherently no silver bullet.“

Bemerkung
Der Begriff „Silver Bullet“ stammt aus der Märchenwelt. Gemäss allgemeiner Ansicht muss ein Werwolf mit einer Silberkugel erschossen werden, sonst stirbt er nicht! Wie ein Werwolf kann auch Software zum Monster werden, wenn das Budget überschritten wird, die Zeitplanung über den Haufen geworfen wurde und alle auf die armen Entwickler drauf hacken!

Gemäss Brooks bestehen die Probleme in den folgenden Bereichen:
- „essence“ : inhärente Probleme der Software, Natur gegebene Probleme der Software
- „accident“ : Schwierigkeiten, die z.B. entstehen, weil die Software später anders eingesetzt wird als ursprünglich geplant, aber nicht Natur gegeben sind. Fehler dieser Kategorie sollten also in Zukunft vermeidbar sein. Dafür könnte also eine silberne Kugel brauchbar sein.

Welche Probleme sind Natur gegeben? Brooks nennt folgende 4 Faktoren
- Komplexität (im Sinne von schwierig, kompliziert)
- Konformität
- Änderungsfreundlichkeit
- Unsichtbarkeit

2.10.1. Komplexität

Software ist komplexer als fast alles was wir bisher konstruiert haben. Uns fehlt also schlicht die Vergleichsmöglichkeit.

Betrachten wir ein einfaches Beispiel:
Sei w ein Wort im Hauptspeicher eines Rechners, mit einer 16 Bit CPU
Jedes der 16 Bits kann die Werte 0 und 1 annehmen.
Somit kann w insgesamt 2**16 verschiedene Zustände annehmen
Wenn wir zwei Worte w1 und w2 haben, dann ist die Anzahl Zustände 2**16 * 2**16 also 2**32
Allgemein:
die Anzahl Zustände eines Systems ist gleich dem Produkt der Anzahl Zustände seiner Komponenten!

Betrachten wir jetzt ein Programm **P**
Das Programm enthält ein 16 Bit Wort **w** zum Speichern eines 16 Bit Wortes x.
Wenn wir nun mit **read(x)** die Variable x einlesen, dann kann x 2**16 Werte annehmen.

Würde das Programm **P** lediglich aus der Anweisung **read(x)** bestehen, dann könnte das Programm 2**16 Zustände annehmen.
In der Realität werden Sie kaum ein so banales Programm antreffen!
Wir können auch beliebig komplexe Programme mit **while** und **for** und **switch** und ... zusammen bauen
Dadurch wird die Komplexität noch wesentlich erhöht: werden while Schleifen endlos durchlaufen? Können Teile des Programmes überhaupt nie ausgeführt werden (ein Optimizer-Problem welches aber auch Hinweise auf Design-Fehler liefert)?

Auch in andern Fachbereichen werden komplexe Systeme beschrieben. In der Physik und der Mathematik betrachtet man chaotische Systeme, die schlicht zu komplex sind, um noch deterministisch beschrieben zu werden. Der genaue Ablauf des Systems ist nicht mehr berechenbar; man muss sich mit Wahrscheinlichkeitsaussagen begnügen.

Aber diese Art der Komplexität unterscheidet sich (scheinbar?) von der Komplexität der Software. Die folge ist nicht nur eine grosse Unsicherheit innerhalb des Entwicklungs-Teams. Auch das Management ist überfordert, da keine genaue Information über den Software-Entwicklungsprozess zur Verfügung steht.

2.10.2. Konformität

Eine Goldgrube soll voll automatisiert werden. Mit Hilfe von einem Software- System soll die Goldgewinnung verbessert und automatisiert werden. Die Software soll die bestehende Abläufe übernehmen. Die Software soll also den bestehenden Abläufen angepasst werden, nicht die Anlage der Software.

Wie würde das Szenario aussehen, wenn an Stelle des angepassten Software- Systems eine völlig neue Goldschürf- Anlage geschaffen würde?
Hoffentlich würden die Bergbau-Ingenieure, die Gold-Experten, Geologen ....und die Software-Ingenieure zusammen sitzen, um eine völlig neuartige Anlage zu entwerfen und später auch zu bauen.
Aber da jedermann den Irrglauben hat, dass Software sich leicht den Gegebenheiten anpassen lässt, werden die Fachexperten wieder eine ähnliche Anlage bauen. Der Software. Ingenieur wird somit wieder die üblichen Probleme haben. Die Komplexität der Software steigt beliebig, weil die Software den gegebenen Hardware und den gegeben Abläufen angepasst werden muss.

2.10.3. Änderbarkeit

Jeder weiss, dass ein Betriebssystem im Laufe von 5 Jahren im Wesentlichen neu geschrieben wird, obschon es offiziell nur gewartet wird. Im Brückenbau würde kaum jemand davon ausgehen, dass die halbe Brücke innerhalb von 10 Jahren neu gebaut werden.

Der Druck auf die Software- Ingenieure, Software neuen Gegebenheiten anzupassen, ist eine Realität und resultiert einmal mehr aus der Annahme, dass sich Software leicht ändern lässt.

Allem Anschein handelt es sich hier um ein Software inhärentes Problem, welches sich nicht lösen lässt.
Software muss dauernd abgepasst werden. Die Funktionalität muss dauernd erweitert werden, obschon nur ein Bruchteil der Funktionen tatsächlich eingesetzt wird!

Gründe warum Software geändert wird:

- Software ist ein Abbild der Realität
  Die Realität verändert sich
- Falls Software nützlich ist, dann besteht das Bestreben sie anzupassen, zu erweitern
- Software lässt sich leichter ändern als Hardware
- Erfolgreiche Software lebt länger als die Hardware

2.10.4. Unsichtbarkeit

Software kann man nicht fühlen!
Was man nicht spürt, bleibt schwer fassbar. Also ist Software und speziell seine Komplexität kaum erfassbar und auch kaum erklärbar.

Teile der Software können wir zwar Visualisieren, aber ein Flow Chart oder ein Datenmodell drückt die Komplexität der Software nur sehr ungenau aus.

2.10.5. Gibt es keinen Silver Bullet?

Gemäss Brooks gab es im Software- Bereich drei grosse Durchbrüche:

- High Level Sprachen
- Timesharing Computer
- Software Entwicklungs- Umgebungen

Brooks untersuchte auch neuere Technologien:

- Objekt Orientierte Techniken
- Experten Systeme
- Künstliche Intelligenz
- Ada

Aber keine dieser neueren Ansätze birgt in sich das Potential die Software Problematik grundlegend zu lösen

Brooks rät zudem folgendes Vorgehen:

- Wann immer möglich soll Kaufsoftware eingesetzt werden
- Rapid Prototyping verbessert vom Ansatz her die Software in frühen Phasen (Anforderungen, Spezifikationen, Designs)
- Inkrementelle Software- Entwicklung, bei der die Software schrittweise entwickelt wird

Brooks vermutet jedoch, dass am meisten mit

- Besserer Schulung
- Förderung von guten Designs und hervorragenden Software Entwicklern

Kommen wir zurück zu harten Zahlen: in den vergangenen Jahren wurde die Produktivität im Software Entwicklungs- Bereich um durchschnittlich 6% pro Jahr gesteigert! Dies entspricht in etwa der Produktivitätssteigerung in andern industriellen Bereichen.

Also: langsam aber sicher werden wir besser. Leider dauert es etwa 12Jahre bis wir die Produktivität verdoppelt haben!

2.11. Zusammenfassung

Wir haben eine erste oberflächliche Betrachtung des Software Prozesses durchgeführt. Speziell haben wir in der Regel Probleme der einzelnen Phasen betont. Zudem haben wir versucht die Frage nach dem Testen in jeder Phase zu beleuchten.

Brooks kritische Bemerkungen zum Software Entwicklungsprozess haben wir im Grunde genommen skizziert und besprochen.

2.12. Aufgaben

- Beschreiben Sie die Projektsituation unter der Voraussetzung, dass der Entwickler mit dem Kunden identisch ist
- Welche Probleme können entstehen, wenn Entwickler und Kunde ein und dieselbe Person sind?
- Ändert sich an Ihren Aussagen etwas, wenn Sie die Methodik wechseln (strukturiert versus OO)?
- Würde es Ihrer Ansicht nach Sinn machen, die Anforderungsphase und die Spezifikationsphase zusammen zufassen, da dadurch die Probleme beim Übergang von der einen zur andern Phase verschwinden würde?
- Welche Dokumentation muss in den einzelnen Phasen des Software Prozesses erstellt werden?
  Requirement Phase
  Spezifikations Phase
  Design Phase
  Implementations Phase
  Integrations Phase
  Wartungs Phase
- Wartungs-Ingenieure verdienen oft weniger als Entwicklungs-Ingenieure. Was falten Sie davon? Begründen Sie Ihre Ansicht!
- Warum gibt es relativ wenig Ablösungen von Software?
- In Ihrem Projekt wurde die gesamte Dokumentation durch einen Headcrash zerstört. Sie müssen aber sofort die Software ausliefern. Welche Auswirkung hat die fehlende Dokumentation auf die Wartung der Software?
- Brooks sagt, dass die Software Entwicklung wegen folgenden Gründen schwierig sei:
  Komplexität,
  Konformität,
  Änderbarkeit,
  Unsichtbarbeit
  Wie wirken sich diese Faktoren in anderen Bereichen aus, z.B. Medizin
- Welches sind Ihrer Ansicht nach wichtige Informatik.- Entwicklungen der letzten Jahre?

- Team-Aufgabe:

Welche Auswirkung auf die Software Entwicklung der Case Study (bereits verteilt) haben die Faktoren „Komplexität“, „Konformität“, „Änderbarkeit“ und „Unsichtbarkeit“ auf das Projekt?

- Literatur-Studium

Vergleichen Sie die zwei Artikel (Brooks und Cox) und nehmen Sie zur Aussage von Cox Stellung, dass Objekte einen „Silber Bullet“ darstellen!

F.P.Brooks : „No Silver Bullet“ abgedruckt in IEEE Computer 20 (April 1987) pp 10-19
B.J. Cox : „There is a Silver Bullet“ in Byte 15 (Oct 19990) pp 209-218
(bereits bestellt)

2. Der Software Prozess und seine Probleme 1

2.1. Kunden, Entwickler und Benutzer 1

2.2. Requirement Phase 2
2.2.1. Requirement Phase Tests 2

2.3. Spezifikations Phase 3
2.3.1. Testen in der Spezifikationsphase 4

2.4. Planungs Phase 5
2.4.1. Testen in der Planungsphase 5

2.5. Design Phase 6
2.5.1. Design Phase Tests 6

2.6. Implementations Phase 8
2.6.1. Testen in der Implementations-Phase 8

2.7. Integrations Phase 9
2.7.1. Integrations Tests 9

2.8. Wartungs-Phase 10
2.8.1. Testen in der Wartungs-Phase 10

2.9. Retirement / Ablösung 11

2.10. Probleme mit dem Software-Prozess 12
2.10.1. Komplexität 12
2.10.2. Konformität 13
2.10.3. Änderbarkeit 13
2.10.4. Unsichtbarkeit 14
2.10.5. Gibt es keinen Silver Bullet? 15

2.11. Zusammenfassung 16

2.12. Aufgaben 17