Software Engineering
im Spannungsfeld Theorie und Praxis

Diplomarbeit

der Philosophisch-naturwissenschaftlichen Fakultät

der Universität Bern

vorgelegt von

Christian Kaufmann

2001

Leiter der Arbeit:

Prof. Dr. Oscar Nierstrasz
Institut für angewandte Mathematik und Informatik, Universität Bern

"You can't write perfect software."
(aus The Pragmatic Programmer)

Zusammenfassung

Die wesentlichen Probleme bei der Softwareentwicklung sind bekannt. Sie wurden in zahlreichen Publikationen beschrieben. Ebenso zahlreich sind die vorgeschlagenen Techniken, Werkzeuge und Methoden zur Lösung dieser Probleme. Trotzdem scheitern zahlreiche Softwareprojekte.

In dieser Arbeit wird nicht eine weitere Methode oder Technik zur Softwareentwicklung definiert. Vielmehr habe ich meine Erfahrungen in einer einfachen Liste von Grundsätzen zusammengefasst. Theorie und Praxis ergänzen diese Grundsätze und zeigen auch ihre Grenzen auf.

Dank

Ich möchte allen Personen danken, die mich bei dieser Arbeit unterstützt haben:

- Prof. Oscar Nierstrasz für die Betreuung und Unterstützung.
- den Firmen und Mitarbeitern, die Zeit und ihr Wissen für die Interviews zur Verfügung stellten.
- Marc Scheuner für die fachliche und Ursula Meier für die sprachliche Korrekturlesung.
- meinem Arbeitgeber B+S Ingenieur AG, dessen Infrastruktur mir jederzeit zur Verfügung stand.
- meiner Familie und Freunden, die mich auch in der Zeit während der Diplomarbeit ertragen mussten.

Inhaltsverzeichnis

1 Einleitung

Softwareentwicklung ist eine sehr junge Wissenschaft. Methoden, Erkenntnisse, die vor fünfundzwanzig Jahren wichtig waren, sind heute längst überholt. So beschreibt zum Beispiel F. Brooks in seinem Buch "The Mythical Man Month" [Brook75a] das Debugging mit Memory Dumps, welches dank schneller Nadeldrucker möglich wurde und welches half, teure Rechenzeit einzusparen. Unvorstellbar in einer Zeit, in welcher einem Entwickler häufig mehrere Rechner zur Verfügung stehen.

Andererseits finden wir im gleichen Buch Feststellungen, die ihre Aktualität und Gültigkeit bis heute nicht verloren haben, wie zum Beispiel "Adding manpower to a late software project makes it later".

Softwareentwicklung ist keine exakte Wissenschaft. Sicher gibt es für Teilbereiche wie etwa Sortieralgorithmen umfassende Theorien, aber schon bei kleinen Softwareprojekten gibt es viele mögliche Lösungen, die einem Problem mehr oder weniger gut gerecht werden. James Rumbaugh umschreibt das in seinem Artikel "What is a method?" wie folgt:

"Softwareentwicklung ist ein kreativer Prozess wie Malen, Schreiben oder Architektur. Beim Malen gibt es Regeln zu Komposition, Farbenlehre und Perspektive, aber damit wird noch niemand zu einem Picasso. Die Auswahl eines Themas und die Gestaltung bleibt Aufgabe jedes Einzelnen."

Softwareentwicklung ist eine breite Wissenschaft. Vergleichen wir zum Beispiel die Steuerung für einen Satelliten und ein Lernprogramm für Kinder. Beides sind Computerprogramme, aber die Anforderungen, Ziele und Schwerpunkte sind völlig unterschiedlich.

1.1 Ziele der Arbeit

Die wesentlichen Probleme bei der Softwareentwicklung sind bekannt. Sie wurden in verschiedensten Publikationen beschrieben. Ebenso zahlreich sind die vorgeschlagenen Techniken, Werkzeuge und Methoden zur Lösung dieser Probleme. Trotzdem scheitern viele Softwareprojekte.

Das Ziel dieser Arbeit ist es nicht, eine weitere Methode oder Technik zu definieren. Vielmehr habe ich meine Erfahrungen in eine einfache Liste von Grundsätzen zusammengefasst. Theorie und Praxis sollen diese Grundsätze bestätigen und ergänzen, sowie ihre Grenzen aufzeigen:

- Der Einbezug der Theorie kommt aus verschiedenen Beispielen der Literatur und wissenschaftlichen Publikationen zum Thema Software Engineering.
- Der Einbezug der Praxis erfolgt mit Hilfe von Interviews und Befragungen, die unter Mitarbeitern in verschiedenen Software Projekten durchgeführt werden.

Das Resultat ist eine Zusammenstellung der Grundsätze, je mit Verweisen auf zugehörige Publikationen, einer Auswertung der Befragungen, sowie praktische Tipps für die Arbeit in Projekten.

1.2 Gliederung der Arbeit

Im Kapitel 2 werden die Gründe für Misserfolge in Softwareprojekten analysiert, klassiert und mögliche Ursachen für die einzelnen Probleme aufgezeigt.

Das Kapitel 3 gibt einen kurzen historischen Rückblick über die Geschichte des Software Engineering, sowie einen Überblick zu einer Auswahl den bekanntesten Methoden zur Softwareentwicklung.

Im Kapitel 4 formuliere ich meine persönlichen Grundsätze, welche ich als wichtig erachte, damit Software Projekte erfolgreich durchgeführt werden können.

Im Kapitel 5 werden die Grundlagen für die Befragungen, sowie eine Zusammenfassung der Resultate der Befragungen dargelegt.

Das Kapitel 6 schlussendlich ist eine Synthese der Kapitel 3 bis 5. Unter Berücksichtigung der Resultate der Befragungen werden die Grundsätze in einer bereinigten Version ausformuliert. Sie werden ergänzt mit Verweisen zu Grundlagen und Publikationen, detailierten Ergebnissen der Befragungen sowie Tipps für die Praxis.

Im Kapitel 7 werden zusätzliche Erkenntnisse, die sich aus den Befragungen, sowie bei der Ausarbeitung des Kapitels 6 ergeben haben, aufgeführt.

2 Bekannte Probleme

Es gibt vermutlich keinen anderen Bereich in der Wirtschaft, in welchem soviele Projekte in irgendeiner Form nicht erfolgreich sind. Die "Standish Group", West Yarmouth, USA hat in einer laufend aktualisierten Studie [Chaos94a] bis heute gegen 30'000 Softwareprojekte untersucht und ist zu folgenden Zahlen gekommen:

- Die untersuchten Projekte wurden in drei Typen aufgeteilt:
  - Typ 1: Das Projekt wurde zeit- und kostengerecht mit allen ursprünglich geforderten Funktionalitäten abgeschlossen.
  - Typ 2: Das Projekt wurde abgeschlossen, aber entweder mit Mehrkosten, zeitlich verzögert oder mit Abstrichen bei den Funktionalitäten.
  - Typ 3: Das Projekt wurde irgendwann im Verlaufe der Entwicklung abgebrochen.
- Der aktuelle Stand der Untersuchung (30'000 Projekte) zeigt, dass nur gerade 16% zum Typ 1 gehören. 52% gehören zum Typ 2 und ganze 31% zum Typ 3.
  Es werden heute also zwar gut zwei Drittel aller Softwareprojekte abgeschlossen, aber nur eine Minderheit der Projekte kann als wirklich erfolgreich betrachtet werden.

Nur wenige dieser Misserfolge werden überhaupt publik. In letzter Zeit waren es vor allem Internetprojekte, bei welchen von völlig falschen Annahmen bezüglich Marktpotenzial ausgegangen wurde. Meistens haben auch weder das Projektteam noch der Auftraggeber ein Interesse daran, dass ein Misserfolg bekannt wird. Gerade bei Aufträgen der öffentlichen Hand ist dies häufig der Fall.

Misserfolge wirken sich auch selten wirklich fatal aus. Vielleicht werden sie auch einfach akzeptiert und/oder nicht als solche wahrgenommen. Das hat vermutlich verschiedene Gründe:

- Neue Generationen von Hardware, Betriebssystemen und Techniken folgten sich in den letzten Jahren so schnell hintereinander, dass Software obsolet wurde, bevor sie überhaupt zum Einsatz kam.
- Nach wie vor besteht ein krasses Missverhältnis bei Angebot und Nachfrage nach Software Ingenieuren und Programmierern, so dass nur in seltenen Fällen eine Konkurrenzsituation besteht. Ein analoges Vorgehen wie zum Beispiel bei einem Architekturwettbewerb ist heute noch unvorstellbar, da jedes Informatikunternehmen auch sonst zu genügend Aufträgen kommt. Somit findet auch unter den Fachleuten nirgends eine Diskussion über die Qualität der Arbeiten statt.
- Auftraggeber sind selten in der Lage, die Qualität der abgelieferten Arbeit frühzeitig und korrekt zu beurteilen, vor allem auch aus Mangel an entsprechend augebildeten Mitarbeitern. Deshalb sind für Auftraggeber persönliche Bekanntschaften häufig ein Kriterium bei der Auswahl eines Unternehmens. Bei Misserfolgen wird dann aus persönlicher Rücksichtnahme verhindert, dass diese überhaupt bekannt werden.
- Seit etwa fünfzehn Jahren ist eine breite Öffentlichkeit mit Computern konfrontiert und hat in dieser Zeit auch weitgehend akzeptiert, dass Software immer fehlerhaft ist. Als wäre es ein Naturgesetz. Wie weit wir das dem grössten Mitspieler im privaten Softwaremarkt zu verdanken haben sei dahingestellt.

Die auftretenden Probleme in Softwareprojekten lassen sich in fünf Kategorien einteilen:

1. Termine werden nicht eingehalten
2. Kosten sind zu hoch
3. Ausgelieferte Software ist fehlerhaft
4. Anforderungen werden nicht abgedeckt
5. Systeme sind nicht wartbar und anpassbar

Fast immer bestehen Abhängigkeiten und gegenseitige Beeinflussung zwischen den einzelnen Faktoren:

- Nicht Einhalten der Termine führt fast immer auch zu hohen Kosten, weil der zeitliche Aufwand auch höher ist.
- Die Kosten können eingehalten werden, dafür sind nicht alle Anforderungen erfüllt oder umgekehrt.

Nicht alle Faktoren in einem Softwareprojekt können gleichzeitig optimiert werden. Es bestehen unter Umständen folgende Abhängigkeiten:

- Kostengünstig und schnell erstellte Software ist schlecht wartbar.
- Zeitgerecht erstellte Software enthält Fehler.
- Software, die sämtliche Benutzeranforderungen zu hundert Prozent abdeckt ist teuer und manchmal schlecht anpassbar.

2.1 Termine werden nicht eingehalten

Auf den ersten Blick handelt es sich hier um einfach messbare Kritierien. Wohl gerade deshalb wird häufig versucht darauf zu verzichten, genaue Termine angeben zu müssen. Borland hat zum Beispiel im Herbst 1999 angekündigt, das Entwicklungswerkzeug Delphi auf Linux zu portieren. Über Release Termine wurde immer nur spekuliert und die Standardantwort von Borland Mitarbeitern in den Newsgroups war "When it's ready." Vermutlich war diese Strategie die Folge von schlechten Erfahrungen mit der Auslieferung von unfertigen Produkten. Zu Delphi 4 gab es insgesamt drei Updatepacks und entsprechend gross war damals auch die Kritik der Benutzer.

Symptome

- Verzögerung bei der Auslieferung der Software.
- Termine werden laufend nach hinten angepasst, häufig mit unklaren oder zweifelhaften Begründungen.
- Verzögerung wegen Abhängigkeiten, zum Beispiel bei Schnittstellen zwischen mehreren beteiligten Firmen.

Mögliche Ursachen

- Auch für erfahrene Entwickler ist es schwierig, den genauen Zeitaufwand abzuschätzen. Die verwendeten Werkzeuge sind immer relativ neu, Es gibt wenig Erfahrungswerte und die Produktivität einzelner Personen ist sehr unterschiedlich.
- Der Mangel an guten Softwareingenieuren führt zu häufigen Stellenwechseln. Die Einarbeitung in ein laufendes Projekt braucht immer Zeit.
- Softwareentwickler sind Optimisten und neigen dazu, ihre Produktivität zu überschätzen. Dieses Phänomen beschreibt schon Brooks in seinem Essay "The mythical man-month"[Brook75a] ziemlich ausführlich.
- Manager von Projekten ignorieren Tatsachen und arbeiten mit unrealistisch engen Zeitplänen, weil sie sich stark an äusseren Einflüssen (Konkurrenz) orientieren.
- Noch immer würden viele Softwarefirmen zusätzliche Leute anstellen, wenn sie welche finden würden. Es wird dann einfach an jenen Projekten gearbeitet, in welchen es am dringensten ist, also häufig wenn die Termine schon vorbei sind.

2.2 Kosten sind zu hoch

Bis heute spielten Kosten eher eine untergeordnete Rolle. Jeder akzeptierte, dass Softwareentwicklung teuer ist. Mit dem Ende des Booms in der IT Branche im letzten Jahr wird der Kostenfaktor je länger je mehr eine wichtigere Rolle spielen. Vermehrt wird dann auch die Forderung nach quantifizierten Abwägungen Kosten/Nutzen auftauchen; etwas, das heute nur selten gemacht wird.

Symptome

- Nachtragsforderungen von ausführenden Firmen.
- indirekte Kosten durch hohe interne Aufwände für Einführung und Schulung.

Mögliche Ursachen

- Bei der Auftragsvergabe standen nur sehr ungenaue Spezifikationen mit viel Interpretationsspielraum zur Verfügung.
- Kosten für zusätzlich nötige Hardware wurden vernachlässigt. Die bei der Entwicklung eingesetzte Hardware war um ein vielfaches leistungsfähiger als die Hardware beim Kunden. Um die Software schlussendlich vernünftig einsetzen zu können, musste die Hardware beim Kunden aufgerüstet oder ersetzt werden. Dies ist heute wohl ein untergeordnetes Problem, Die Entwicklungen bei der Hardware ist zwar immer noch rasant, die zusätzlich mögliche Rechenleistung wird aber für die meisten Projekte gar nicht verwendet. Am ersten tritt das Problem noch in grösseren Unternehmen auf, wo mehreren hundert oder tausend Rechnern im Einsatz ist und der natürliche Wechsel einer ganzen Computergeneration noch entsprechend lange dauert.
- Kosten für Einführung und Personalschulung wurden vernachlässigt. Solche Kosten werden häufig gar nicht ausgewiesen sondern sie verschwinden im internen Aufwand und äussern

sich durch tiefere Produktivität der Mitarbeiter. Dabei kann der Einsatz von neuen Technologien oder Projekte in neuen Fachgebieten mit sehr grossem Einarbeitungsaufwand verbunden sein.

- Kosten für Datenerfassung und Datenkonvertierung wurden vernachlässigt. Dies ist ein recht häufiges Problem, da bei vielen Businessapplikationen dieser Aufwand bedeutend grösser ist, als die Softwareentwicklung selber. Es besteht hier auch ein starker Zusammenhang mit der Brauchbarkeit einer Software. Falls zum Beispiel wichtige Basisdaten aus Kostengründen nicht erfasst werden, kann die Software die gesteckten Anforderungen gar nicht erfüllen.

2.3 Ausgelieferte Software ist fehlerhaft

Der "Vorteil" dieser Art von Problemen ist, dass sie meistens schnell sichtbar werden. Der Nachteil ist, dass der Anwender verunsichert wird und gerade in der Phase, in welcher er sich in etwas Neues einarbeiten muss, noch mit zusätzlichen Problemen konfrontiert wird.

Symptome

- Die Software kann nicht installiert werden.
- Die Software liefert unerwartete oder nicht verständliche Fehlermeldungen.
- Die Software stürzt ab.
- Es finden ungewollte Manipulationen an Daten statt. Dies ist wohl die gravierendste Art von Problemen bei fehlerhafter Software, da solche Fehlmanipulationen irreversibel sein können. Und zuverlässige Backups fehlen häufig bei kleineren oder mittleren Unternehmen.

Mögliche Ursachen

- Es wurden keine Tests durchgeführt. Jeder Entwickler weiss, dass getestet werden soll, aber Tests sind uninteressant, zeitintensiv ohne sichtbares Resultat.
- Es fehlt eine saubere Versionskontrolle. Sobald mehr als ein Entwickler am gleichen Projekt arbeiten ist es zwingend nötig ein System einzusetzen, das garantiert, dass Änderungen und Erweiterungen immer nur auf der aktuellsten Version des Sourcecodes gemacht werden und, sobald diese fehlerfrei ist, auch allen Entwicklern wieder zur Verfügung gestellt wird.
- Updateprozesse beim Endanwender sind nicht geplant und kontrolliert. Erstinstallationen sind immer einfacher als die Installation von Updates. Bei Updates treten häufig Fehler auf, weil in Datenstrukturen etwas geändert wurde, ohne dass der dazugehörende Updateprozess beim Anwender geplant und durchgeführt wurde.
- Betriebssysteme sind heute so komplex, dass kaum zwei Computer die gleiche Betriebsumgebung anbieten. Verschiedene Software auf dem gleichen Computer beeinflusst sich gegenseitig, vor allem über "shared libraries", bei welchen es häufig zu Konflikten mit verschiedendlichen Versionen kommen kann. Das letzte Problem tritt vor allem bei den Betriebssystemen der Familie Windows von Microsoft auf, wo die Grenze zwischen Betriebssystem und Applikationen sehr fliessend ist und viele Applikationen stark miteinander gekoppelt sind.

- Es fehlt häufig eine solide Grundausbildung bei den Entwicklern, die dazu beitragen würde, dass Projekte geplant und strukturiert abgewickelt werden. Noch immer sind viele Softwareentwickler Quereinsteiger, die sich ihr Wissen irgendwie nebenbei angeeignet haben, jeweils genau für ein gerade anstehendes Problem. Es fehlt ihnen eine Gesamtsicht und ein Grundwissen, das von den gerade aktuellen Technologien unabhängig ist.

2.4 Anforderungen werden nicht abgedeckt

Vermutlich einer der häufigsten Gründe für missglückte Projekte. Der Misserfolg wird nicht unbedingt sofort sichtbar, da sich Probleme der fehlenden Schulung und fehlender oder falsch realisierter Funktionalitäten überlagern. Erst mit der Zeit treten die Mängel in der Realisation als solche in Erscheinung.

Symptome

- Die erstellte Software wird wenig oder nur teilweise verwendet. In extremen Fällen kommt sie gar nicht zum Einsatz.
- Eine erwartete Zeiteinsparung oder Produktivitätssteigerung tritt nicht ein. Gewisse Arbeiten können zwar einfacher erledigt werden, aber demgegenüber werden zusätzliche, teilweise unnötige Arbeiten verursacht.
- Die Einführung der Software erfordert viel Zeit und einen hohen Schulungsaufwand bei den Anwendern.

Mögliche Ursachen

- Die Prozesse, bei welchen die Software zum Einsatz kommt, wurden zuwenig genau analysiert. Es fehlen zum Beispiel klare Ziele, wo Verbesserungen erreicht werden sollen (Geschwindigkeit im Ablauf, Qualität, Umfang).
- Bei der Ausarbeitung der Spezifikationen waren die Kontaktpersonen des Auftraggebers zuwenig gut mit den zu lösenden Problemen vertraut.
- Es wurde versucht, ein Problem zu umfassend bis ins kleinste Detail mit einer neuen Software zu lösen. Das so entstandene System wurde unübersichtlich, es fehlt eine klare Aufteilung zwischen wichtigen und häufig gebrauchten Funktionen und Spezialfunktionen, welche selten oder gar nie zum Einsatz kommen.

2.5 Systeme sind nicht wartbar und anpassbar

Auch diese Art von Misserfolg von Softwareprojekten wird selten sofort sichtbar. Sie zeigt sich erst im Lauf der Zeit, in welcher ein System oder eine Applikation im Einsatz ist. Die laufend nötigen kleineren Anpassungen werden immer aufwendiger und führen immer häufiger zu unerwünschten Seiteneffekten. Irgendwann wird der Punkt erreicht, an welchem man beschliesst, nichts mehr an der Software zu ändern. Weitere nötige Anpassungen führen dann zum vollständigen Ersatz der bestehenden Software.
Hunt bezeichnet dieses Phänomen als "Software Entropy". Er schreibt dazu: "While software development is immune from almost all physical laws, entropy hits us hard."[Hunt99a] Entropie ist ein physikalisches Mass der Unordnung und kann in einem geschlossenen System nur zunehmen. Um die Entropie in einem System zu verringern braucht es zusätzliche Energie von aussen.

Symptome

- Schon kleinste Anpassungen oder Ergänzungen an der Software sind sehr aufwendig, dauern lange und sind teuer.
- Eine Portierung auf neuere Betriebssystemversionen ist schwierig oder gar nicht möglich.
- Grundsätzliche Erweiterungen (zum Beispiel Einplatz -> Mehrplatz) bedeuten, dass die ganze Software vollständig neu geschrieben werden muss.

Mögliche Ursachen

- Den beteiligten Entwicklern fehlt eine gute Grundausbildung in Software Engineering. Wegen der nach wie vor herrschenden Goldgräberstimmung sind immer noch viele Hobbyinformatiker tätig. Auf einer Website für Softwareentwickler gab es kürzlich folgende Umfrage "Setzen Sie Design Patterns ein?" Die Aufteilung der zirka 650 Antworten war: 15% Ja, 13% Nein, 72% Was ist ein Design Pattern ? Das mehr als zwei Drittel der Personen bei einer freiwilligen Befragung dazu stehen, dass sie nicht wissen was ein Design Pattern ist zeigt, dass eine Grundausbildung wirklich häufig fehlt.
- Die Erfahrung mit Langzeitprojekten fehlt fast vollständig, da es bei kleineren und mittleren Systemen häufig effizienter ist, bei Änderungswünschen die ganze Software neu zu schreiben und dabei neue Methoden, Tools und Libraries mit all den zusätzlichen Möglichkeiten zu nutzen.
- Aus Termingründen werden Programme so fertiggestellt, dass sie "einfach irgendwie laufen". Dies geschieht häufig auf Kosten einer sauberen Implementation. Folgen sind "Quick and dirty" Funktionen, duplizierter Code und schlechte Dokumentationen.
- Selten findet bereits bei der ersten Entwicklung ein Code- und Design-Review statt. Diese Art von Qualitätskontrolle kommt heute erst zum Einsatz, wenn beispielsweise ein neues Team ein bestehendes Projekt übernimmt und einen Überblick über vorhandene Teilsysteme und Funktionen erlangen muss.

3 Lösungsvorschläge

Ein kurzer Abriss über die Geschichte des Software Engineering, soll ein besseres Verständnis dafür geben, wo wir heute stehen und welche Methoden noch, wieder oder noch nicht verwendet werden.

Weitere Abschnitte in diesem Kapitel beschreiben die wichtigsten heute eingesetzten Methoden oder Methodengruppen, ihre Charakteristiken, ihre Stärken und Schwächen.

3.1 Geschichte Software Engineering

Die Geschichte des Software Engineering lässt sich nicht als einfacher zeitlicher Ablauf formulieren. Vielmehr ist es das Zusammenspiel von verschiedenen parallelen Entwicklungen, die gegenseitige Beeinflussung von theoretischen Erkenntnissen, technischer Möglichkeiten und viel Marketing.

3.1.1 Die Anfänge

In den Anfängen der Computer gab es keine speziellen Techniken zur Softwareentwicklung. Mit Fortran, Cobol und Algol entstanden Ende der Fünfziger Jahre die zweite Generation von Programmiersprachen. Diese brachten eine gewisse Abstraktion von der Programmierung in Maschinencode (erste Generation).
Algol brachte zudem die Idee der strukturierten Programmierung mit den klassischen Programmierstatements der Verzweigung (IF THEN ELSE) und der verschiedenen Kontrollschleifen (FOR, WHILE, REPEAT UNTIL), die Aufteilung des Codes in Blöcke und die Einführung von lokalen Variablen. Damit verbunden ist eigentlich die Methode, ein Problem in zahlreiche Teilprobleme aufzuteilen, eine alte, einfache aber noch heute sehr effiziente und wichtige Methode. Algol ist auch der Vorläufer zahlreicher bekannter prozeduraler Programmiersprachen wie C und Pascal.

3.1.2 Die Softwarekrise

Die Grösse und Komplexität von Programmen wuchs in den sechziger Jahren sehr schnell, die Fehlerhäufigkeit war gross und es kam zu einer sogenannten Softwarekrise. In den Jahren 1968/69 fand unter dem Regime der NATO Tagungen zum Thema Softwareentwicklung statt. In seinem Vortrag[McI69a] stellte McIlroy den Bezug zwischen Software und industrieller Produktion her. Er postulierte folgende Punkte:

- Jede reife Industrie basiert auf Komponenten. Komponenten sind austauschbare Module mit klar definierten Schnittstellen.
- Erst der Einsatz von Komponenten machen eine Massenproduktion möglich.
- Erst Systeme, welche aus Komponenten zusammengebaut werden, sind beherrschbar und wartbar.

Später erfand McIlroy bei den Bell Labs die UNIX pipes, welche bis heute das meisteingesetzte Komponentensystem sind.

3.1.3 Erste Methoden für Software Engineering

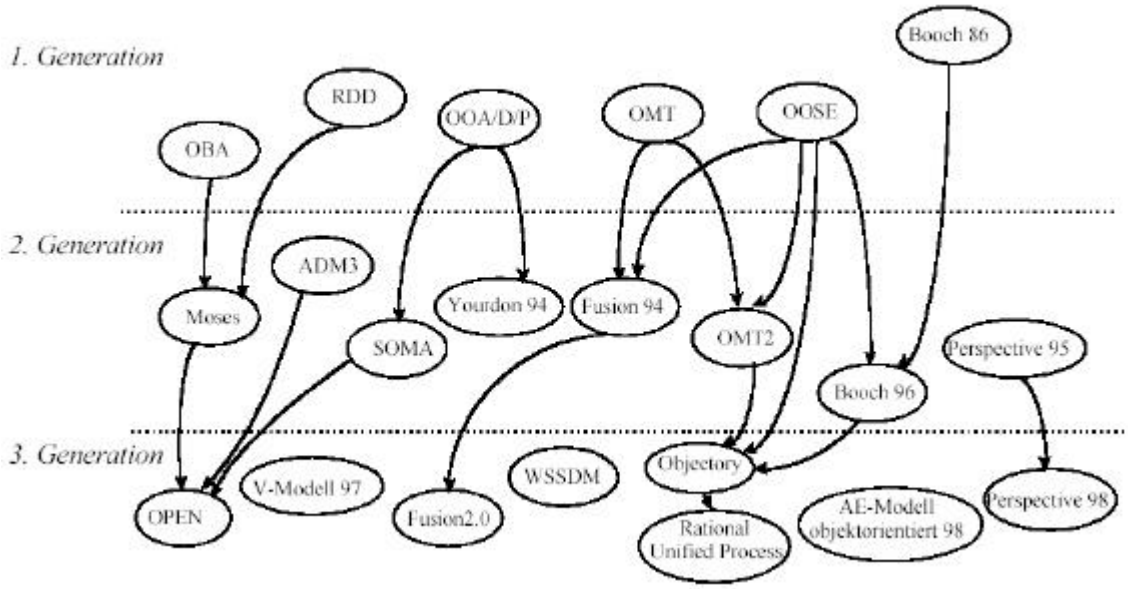
Abgeleitet von den Methoden der klassischen Ingenieurwissenschaften wie zum Beispiel Bauingenieur Methoden, entstand das Phasenmodell (siehe Abschnitt 3.2). Die Methode geht davon aus, dass Softwareprojekte wie zum Beispiel Bauprojekte in Entwurf/Design und Ausführung/Implementation aufgeteilt werden können. Dadurch erreicht man, dass die Phase der Ausführung zeitlich und finanziell abschätzbar ist.
Methoden, welche auf dem Phasenmodell beruhen sind sehr leistungsfähig, aber auch sehr langsam und bürokratisch, da man einen sehr genauen Prozess befolgen muss.

3.1.4 Der PC und die globale Verbreitung des Computers

Das Aufkommen der PC's (Personal Computer) Anfang der Achtziger Jahre führte zu drastischen Änderungen in der Softwareentwicklung. Einerseits verfügte diese neue Generation von Computern im Vergleich zu grossen Mainframes nur über sehr wenig Ressourcen (Hauptspeicher / Festplatten), so dass Programme gezwungenermassen sehr klein und übersichtlich sein mussten. Auf der anderen Seite schuf die weite Verbreitung der Computer eine neue grosse Gruppe von Hobbyprogrammierern, welche etwa mit den gleichen Methoden vorging, wie die Computerspezialisten der Fünfziger und Sechziger Jahre, nämlich keiner.
Erst mit der Zunahme der Leistungsfähigkeit der PC's und damit der Zunahme der Komplexität der Applikationen erkannte man, dass methodisches Vorgehen nötig ist. Die PC Welt übernahm die Methoden, welche in der Softwareentwicklung auf Grossrechnern schon lange im Einsatz waren.

3.1.5 Object Oriented Programming

Objektorientierte Programmiersprachen und damit objektorientierter Softwareentwurf begann Ende der Achtziger Jahre. Es wurde versucht, die zunehmende Komplexität von Software (graphische Oberflächen und parallel laufende Programme), mit objektorientieren Programmiersprachen in den Griff zu bekommen.
Die erste Generation objektorientierter Vorgehensmodelle orientierte sich stark an einzelnen Sprachen wie Ada, Smalltalk und Eiffel oder Methoden wie etwa Booch, RDD oder OOSE. Die zweite Generation umfasste neben der eigentlichen Entwicklungssicht oft bereits weitere komplementäre Sichten wie etwa die Qualitätssicherung oder das Projektmanagment. Bei der Entwicklung der Vorgehensmodelle der dritten Generation treten anstelle einzelner Autoren typischerweise Konsortien, Werkzeughersteller oder grosse Organisationen in Erscheinung. [Noa99]



Genealogie objektorientierter Vorgehensmodelle

Um OO Software zu entwerfen wurden verschiedene Vorschläge für Notationen gemacht, welche schlussendlich Mitte der Neunziger Jahre in UML (Unified Modeling Language) zusammgeführt wurden. Der eigentliche Prozess bei der Softwareentwicklung wurde während der Entstehung von UML nicht näher untersucht oder hinterfragt. Das Phasenmodell kam weiterhin zum Einsatz, vor allem auch weil man überzeugt war, mit UML nun eine Möglichkeit zu haben, Software vor der Implementation umfassend und exakt beschreiben zu können.

3.2 Der klassische Softwarezyklus (Phasenmodell)

Die Erstellung einer Software wird in Phasen eingeteilt. Mit der nächsten Phase wird jeweils erst begonnen, wenn die vorherige Phase abgeschlossen ist.
Die Phasen sind:

- Erheben der Anforderungen: Sammeln und Festhalten der Anforderungen. Der Kunde soll hier sehr genau und abschliessend angeben, was die Software schlussendlich leisten soll.
- Analyse und Entwurf: Analyse der Anforderungen und Entwurf der Software. Diese Phase wird als wichtigster Schritt einer Softwareentwicklung angeschaut. Ziel ist es, die ganze Applikation abschliessend zu beschreiben, analog zu den Ausführungsplänen eines Bauwerkes.
- Codieren und Testen: Implementation der Applikation aufgrund des Entwurfs. Da ja der Entwurf sehr umfassend und abschliessend sein soll, kann diese Arbeit sehr schnell und kontrolliert ausgeführt werden.
- Wartung: Unterhalt der Software, Fehlerkorrekturen, Anpassungen und Erweiterungen.

Für Projekte mit sehr klarem Leistungsumfang und hohen Anforderungen an Zuverlässigkeit und Sicherheit (zum Beispiel Steuerungen von Kraftwerken) erwies sich diese Methode als sehr leistungsfähig.

Der grösste Nachteil des phasenweisen Vorgehens ist, dass zu Projektbeginn sämtliche Anforderungen klar definiert sein müssen. Zudem erhält der Kunde erst nach Abschluss der gesamten Arbeit eine einsatzfähige Software.

Die Schwerfälligkeit dieser Methoden brachte ihnen den Namen der "Heavyweight methods" oder gar "Monumental methods" ein.

3.3 Das Spiral Modell

Das Spiralmodell wurde ursprünglich von Boehm[Boe88a] vorgeschlagen. Es handelt sich um einen evolutionären Softwareprozess. Während der einzelnen Iterationen entstehen jeweils unterschiedliche Resultate. Eine mögliche Abfolge beginnt mit der Produkte Spezifikation, gefolgt von einem oder mehreren Prototypen, dann nach und nach vollständigere Versionen der Software. Jede Iteration wird immer in die gleichen Schritte aufgeteilt, normalerweise zirka in drei bis sechs Schritte. Eine mögliche Aufteilung ist:

- Kommunikation mit Kunden: Organisation der Kommunikation mit dem Kunden, um die nächste Iteration der Entwicklung beginnen zu können.
- Planung: Aufstellen des Zeitplans und Planung der nächsten Aktivitäten und der dazu benötigten Ressourcen.
- Risikoanalyse: Abschätzung der möglichen Risiken der Planung bezüglich der zu verwendenden Technik und der gewählten Organisation.
- Engineering, Konstruktion und Release: Erstellen der Applikation, Tests, Installation beim Kunden und Ausbildung der Benutzer.
- Evaluation durch Kunde: Feedback des Kunden zu den in der letzten Iteration neu erstellten Applikationen.

Das Spiralmodell eignet sich vor allem für die Entwicklung sehr grosser Projekte. Mit jeder Iteration erhalten Kunde und Entwickler ein besseres Verständnis für die Probleme und können auf mögliche Risiken und Änderungen im Umfeld reagieren. Prototyping wird hier als Mechanismus zur Risikoverminderung eingesetzt und zwar nicht nur ganz zu Beginn eines Projektes sondern wiederholt in jeder neuen Iteration.

3.4 Object Oriented Design

3.4.1 Allgemeine Prinzipien

Die wichtigsten Prinzipien in welchen sich Object Oriented Design (OO) von herkömmlichen Design Methoden unterscheidet, sind die folgenden:

- Kapselung: Daten und Operationen werden in Objekten zusammengefasst. Gleichartige Objekte werden durch Klassen beschrieben. Grundsätzlich sollen Daten nur über das Ausführen von Operationen der Objekte manipuliert werden können.
- Vererbung und Polymorphie: Ein wesentliches Schlüsselkonzept der Objektorientierung ist die Möglichkeit der Vererbung von Eigenschaften und Verhalten von einer Klasse auf eine andere. Dieses Prinzip stellt gleichzeitig die Grundlage für eine bessere und leichtere Wiederverwendbarkeit von Softwarebausteinen dar. Das Grundprinzip der Vereerbung ist gleichzeitig auch die Voraussetzung für polymorphes Verhalten, das heisst die Vielgestaltigkeit von Objekten, je nach verwendetem Kontext.

Neben diesen Grundprinzipen die bei allen OO Methoden und Programmiersprachen vorkommen, sind von verschiedenen Personen weitere Prinzipien postuliert worden:

- Design by Contract: 1985 entwarf Bertrand Meyer die objektorientierte Programmiersprache Eiffel, in welcher er "Design by Contract"[Mey93a] als integralen Bestandteil einbaute. Die Idee bei "Design by Contract" ist, dass zwischen aufrufender Klasse (Client) und aufgerufener Klasse (Supplier) Verträge abgeschlossen werden:
  - Klasseninvarianten: Eine Klasseninvariante garantiert, dass eine bestimmte Bedingung für alle Instanzen einer Klasse zu jeder Zeit erfüllt ist. Zum Beispiel ist die Anzahl der Elemente in einem Stack immer grösser oder gleich Null.
  - Vorbedingungen (Preconditions): Beim Aufruf einer bestimmten Methode garantiert eine Vorbedingung, dass bestimmte Voraussetzungen erfüllt sind. Zum Beispiel darf ein Stack nicht leer sein, wenn daraus ein Element entfernt werden soll.
  - Abschlussbedingungen (Postconditions): Nach dem Aufruf einer bestimmten Methode, garantiert eine Abschlussbedingung, dass gewisse Dinge erfüllt sind. Zum Beispiel wird ein Stack nie leer sein, nachdem ein Element eingefügt wurde.

- Responsibility-Driven Design (RDD): Responsibility-Driven Design (RDD) wurde von Wirfs-Brock[WiWi90a] eingeführt. Der Grundgedanke beim RDD ist es, die Klassen über ihre Verantwortlichkeiten anstatt über die enthaltenen Daten zu finden. Bei einer Klasse handelt es sich also um eine Abstraktion des Verhaltens. Wichtig ist, was Objekte einer bestimmten Klasse *tun* und nicht was sie *enthalten*.

3.4.2 Unified Modeling Language (UML)

Mit Hilfe von UML versuchten die Verfasser der verschiedenen Prinzipen eine gemeinsame Notation zu definieren. Die wichtigsten Vorgänger von UML sind Booch, OMT und OOSE. Die Autoren von UML bestätigen zwar die Wichtigkeit eines Prozesses, sagen aber zugleich, dass dieser sehr stark von Firmen, Kultur und zu lösendem Problem abhängig sei. Zudem sei die Wahl des richtigen Prozesses auch stark vom Problemgebiet, der verwendeten Technologie (Tools / Programmiersprache) und der Erfahrung der beteiligten Leute abhängig.

Unter den Entwicklern, die mit UML arbeiten, haben sich verschiedene Formen von *Use Case gesteuerten, Architektur zentrierten, iterativen* und *inkrementellen* Prozessen etabliert.

3.4.3 OOSE-Methode (Jacobson)

Jacobson hat in seiner OOSE-Methode ein Prozessrahmenwerk (Objectory-Process) beschrieben, welches Fowler in einem Entwicklungsprozess detaillierter ausformuliert hat [Fow97a]. Im Wesentlichen besteht der Prozess aus:

- Anforderungsphase
- Festlegungsphase
- Erstellungsphase
- Übergabephase

Diese Teilschritte entsprechen ziemlich genau dem klassischen Phasenmodell. Im Gegensatz zum linearen Entwicklungsprozess soll aber bei der OOSE-Methode der Prozess über mehrere Iterationen erfolgen.

3.4.4 Catalysis

Catalysis ist ein Ansatz zur systematischen Entwicklung komponentenorientierter Systeme basierend auf dem Standard von UML. Catalysis entstand 1992 und wurde seither von den beiden Autoren Desmond D'Souza und Alan Wills kontinuierlich weiterentwickelt[SW98a]. Schwerpunkte in Catalysis sind unter anderem:

- Komponentenbasierte Softwareentwicklung: Komponenten können sowohl einzelne Objekte als auch ganze Subsysteme sein. Catalysis definiert, wie Schnittstellen, unabhängig von der Implementation, sauber beschrieben werden können. Catalysis bietet Unterstützung beim Entwurf von Komponentensystemen und Schnittstellen um sicherzustellen, dass die Komponenten einfach austauschbar sind und verbunden werden können.
- Softwarearchitektur: Catalysis setzt einen Schwerpunkt auf die Konzeption der Softwarearchitektur eines Systems. Catalysis unterstützt zudem ein Verständnis der fraktalen Natur einer Architektur: Verfeinerungen führen stets zu denselben Konzepten nur auf anderen Abstraktionsebenen.

3.5 Light Methods

Als Reaktion auf die "Heavy weight methods" entstand in den letzten Jahren eine neue Gruppe von Methoden. Im Wesentlichen sind diese ein Kompromiss zwischen einem zu restriktiven bürokratischen Prozess und keinem Prozess. Gemäss Fowler zeichnen sich "Light Methods" vor allem durch folgende zwei Eigenschaften aus:

- *Lightweight Methoden sind anpassungsfähig und nicht vorausschauend*. Veränderungen sind hier Bestandteil des Prozesses. Im Laufe der einzelnen Entwicklungsphasen wird der Prozess ständig überprüft und angepasst.
- *Lightweight Methoden orientieren sich am Menschen und nicht am Prozess*. Die Eigenheiten der menschlichen Natur werden explizit in den Prozessen berücksichtigt. Die daraus folgende bessere Motivation der Mitarbeiter, führt zu selbstständigerem Denken und mehr Freude beim Entwicklen von Software.

Fowler gibt in seiner Publikation "The New Methodology"[Fow01a] einen umfassenden Überblick über verschiedenste Light Methods. Der bekannteste Vertreter heute ist sicher "Extreme Programming" (XP). Weniger bekannt, aber neben XP eine sehr interessante Methode oder besser Methodenfamilie ist die "Crystal Family" von Alistair Cockburn. Cockburn stellt bei seinen Methoden als einziger konsequent die beteiligten Personen in den Mittelpunkt.

3.6 Extreme Programming

Extreme Programming ist der heute bekannteste Vertreter der "Lightweight methods". Bei Extreme Programming (XP) handelt es sich um ein Prozessmodell zur objektorientierten Softwareentwicklung. Es eignet sich vor allem für kleinere Projekte mit unklaren und sich ändernden Anforderungen. Dabei werden sowohl an die Entwickler als auch an den Auftraggeber (Kunden) hohe Anforderungen gestellt.

XP basiert auf vier Grundwerten: Kommunikation, Feedback, Einfachheit und Mut. Dazu kommen etwa ein Dutzend Regeln und Praktiken. Beispiele dazu sind:

- Kunde vor Ort: Um eine möglichst einfache und direkte Kommunikation mit dem Kunden zu ermöglichen, sollte immer ein Endanwender im Projektteam integriert werden. Nach jeder Iteration erhält der Kunde ein lauffähiges Produkt und kann darauf Einfluss nehmen (Anpassung der Anforderungen).
- Kleine Versionen: Jede neue Version soll so klein wie möglich sein, aber zugleich die für das Gesamtprojekt wichtigsten Erweiterungen enthalten. Kleine Versionen garantieren schnellen Userfeedback und ein einfacheres Zeitmanagement des Projektes.
- Pair Programming: Zwei Entwickler arbeiten am selben Terminal. Einer tippt, der andere überprüft, denkt mit oder denkt voraus. Damit werden eine Menge Fehler schon gesehen und korrigiert, bevor das Programm erstmals getestet wird. Auf den ersten Blick kommt man zwar nur halb so schnell vorwärts, da zwei Personen das machen, was sonst einer allein macht. Auf den Gesamtaufwand der Entwicklung inklusive Tests ist man aber wieder gleich schnell, da viele Fehler erst gar nicht entstehen.

- Tests: Für jedes Modul eines Softwareprojektes soll zuerst ein Test geschrieben werden. Dadurch wird der Entwickler gezwungen sich mit dem Problem auseinanderzusetzen, bevor er mit der Implementierung beginnt. Bei jedem Release müssen sämtliche Tests ausgeführt werden und zu 100% positiv verlaufen.
- Refactoring: Neben Tests ist Refactoring eines der wichtigsten Merkmale von XP. Mit Refactoring meint man die Anpassung des Designs und der Implementation ohne Veränderung der Funktionalität. Da bei jeder Version jeweils nur das nötigste implementiert werden soll, ist die Gefahr gross, dass bei Erweiterungen das Design mehr und mehr nicht mehr den Bedürfnissen des Kunden entspricht. Anpassungen sind hier also dauernd nötig.

Eine Gewichtung der einzelnen Regeln und Praktiken ist schwierig. Sie können auch einzeln eingesetzt werden, jedoch entfalten sie laut Kent Beck erst gemeinsam die volle Wirkung.

Schlussendlich befolgt man einen iterativen Prozess, welcher nach einem sehr strengen Muster abläuft:

- Anforderungen aufnehmen: Dies geschieht mit Hilfe von "User Story Cards", wo der Kunde eine bestimmte Funktionalität beschreibt. Die Entwickler legen darauf hin die Zeit fest, in der diese Anforderungen erfüllt werden können. Die Rollen können auch vertauscht werden, das heisst, der Kunde legt den Zeitrahmen fest und der Entwickler gibt an, was in dieser Zeit in einem bestimmten Modul alles für Features realisiert werden können. Durch Gruppieren der User Story Cards werden Umfang und Zeitpunkt der einzelnen Releases festgelegt. Dieser ganze Vorgang wird auch Planungsspiel genannt.
- Entwicklungsphase: In der Entwicklungsphase durchläuft der Prozess mehrere Iterationen. Am Ende jeder Iteration steht ein lauffähiger Relase der Software zur Verfügung. Jede Iteration besteht aus Implementierung (Pair Programming), Refactoring und Tests. Am Ende einer Iteration wird der gesamte Relaseplan überprüft, die Inhalte oder der Zeitrahmen nötigenfalls angepasst.

XP ist nicht in jedem Fall gut geeignet. Grosse Teams, nicht vertrauenswürdige Kunden und Technologien können den Einsatz von XP verunmöglichen.

Detaillierte Informationen zu Extreme Programming sind im Buch von Kent Beck[Beck99a] oder auf zahlreichen Websites zu finden.

3.7 Rapid Application Development (RAD)

Wegen des Aufkommens von einfach zu handhabenden Programmier-Frameworks hat RAD heute in der Softwareentwicklung eine zentrale Bedeutung. Komplexe Anwendungen lassen sich sehr einfach mit einem visuellen Editor erstellen und das Framework generiert den dazu nötigen Programmcode. Es handelt sich um keine eigentliche Methode, aber die eingesetzten Werkzeuge beeinflussen das Vorgehen bei der Softwareentwicklung sehr stark. Die zentralen Elemente welche jedes RAD-Werkzeug enthält, sind:

- Komponentenbibliothek: Funktionalitäten, vorallem für GUI Oberflächen, aber auch für Datenbankzugriffe und Internetapplikationen werden in Komponenten zusammengefasst. Das Erstellen einer Applikation umfasst im Wesentlichen das Verbinden und Konfigurieren von solchen Komponenten.
- Entwicklungsumgebung: Diese umfasst einen visuellen Editor und einen Texteditor, einen integrierten Compiler und in den meisten Fällen auch einen Debugger.
- Programmiersprache: Bei der verwendeten Programmiersprache kann es sich von einer einfachen Skriptsprache bis hin zu einer voll objektorientierten Sprache handeln.

Beispiele für solche Entwicklungsumgebungen sind:

- MS Access von Microsoft, Programmiersprache Visual Basic
- Delphi von Borland, Programmiersprache Object Pascal
- JBuilder von Borland, Programmiersprache Java
- VisualAge von IBM, für verschiedene Programmiersprachen

Auf der einen Seite ermöglicht der Einsatz solcher Werkzeuge, mit sehr geringem Initialaufwand sehr schnell relativ komplexe Applikationen zu erstellen. Auf der anderen Seite wirkt sich genau das häufig negativ auf ein methodisches Vorgehen aus. Marco Cantù hat das am Beispiel von Delphi beschrieben[Cantu99a].
Ein langfristiger und nachhaltiger Einsatz von RAD Werkzeugen ist nur möglich, wenn man mit den Grundlagen der OO-Programmierung, auf welcher diese Werkzeuge aufbauen, vertraut ist. Auch lässt sich ein gutes Gesamtkonzept für einzelne Applikationen oder ganze Systeme nie mit Hilfe von solchen Werkzeugen erstellen sondern nur mit der Unterstützung von objektorientierten Design Methoden.

4 Grundsätze zum Erfolg

4.1 Projekte scheitern trotzdem

Obwohl die Probleme bekannt und klassifiziert sind (siehe Kapitel 2) und in den letzten dreissig Jahren zahlreiche Methoden vorgeschlagen wurden (siehe Kapitel 3), zeigen auch die aktuellsten Zahlen des Chaos Reports der Standish Group[Chaos94a], dass nach wie vor eine grosse Zahl von Projekten nicht erfolgreich abgeschlossen werden.

Die Erfahrung zeigt, dass es viel einfacher ist zu kritisieren und festzustellen, das etwas falsch gelaufen ist, als Vorgaben und Hinweise zu geben, wie man es besser machen könnte.

Mit den folgenden Grundsätzen möchte ich versuchen, einen Beitrag in dieser Richtung zu leisten. Es ist dabei schwierig, einen Mittelweg zwischen allgemein gültigen und damit wenig hilfreichen Aussagen auf der einen Seite und sehr konkreten und deshalb nicht in jedem Fall richtige Forderungen zu finden.

Die Grundsätze basieren auf persönlichen Erfahrungen und Beobachtungen, sowie dem Studium verschiedenster Arbeiten zum Thema Softwareentwicklung. Es handelt sich dabei um eine Positivliste, dass heisst, wenn diese Forderungen erfüllt sind, hat man eine sehr grosse Sicherheit, dass ein Projekt erfolgreich abgeschlossen wird. Ein Projekt kann auch erfolgreich abschliessen, wenn vielleicht eine, maximal zwei dieser Forderungen nicht erfüllt sind.

In den folgenden Kapiteln werden die Grundsätze mit Hilfe von Resultaten einer Befragung, sowie anderen Publikationen und Ansichten detaillierter und an konkreteren Beispielen behandelt. Dabei handelt es sich nicht um eine abschliessende Betrachtung, sondern viel mehr um eine Momentaufnahme, des aktuellen Standes der Forschung und Entwicklung zu diesem Thema.

Die Formulierung der Grundsätze im aktuellen Kapitel entspricht der ursprünglichen Form, wie sie in den Interviews den befragten Personen vorgelegt wurden. Vereinzelte Aussagen darin erwiesen sich als falsch oder unverständlich formuliert. Im Kapitel 6, in welchem die einzelnen Grundsätze detailliert behandelt werden, habe ich deshalb jeweils eine leicht überarbeitete Version der Grundsätze formuliert.

4.2 Meine Grundsätze für erfolgreiche Projekte

4.2.1 Grundsatz 1: Einbezug Benutzer

Der Benutzer muss zu jedem Zeitpunkt vollständig in die Projektabwicklung integriert sein. Spezifikation und Diskussion muss in der Sprache des Benutzers stattfinden. Dazu geeignete Hilfsmittel sind Skizzen von Bildschirmmasken und Reports, einfache Ablaufdiagramme, Checklisten und Prototypen. Es empfiehlt sich für jedes Projekt einen "Key User" zu bestimmen, der einerseits mit der Problemstellung und den Abläufen sehr gut vertraut ist und der auch in der Lage ist Wünsche und Anpassungen verständlich zu formulieren.

Motivation
Dass der Einbezug der Benutzer eine zentrale Bedeutung hat, lässt sich direkt aus dem Problem "Anforderungen werden nicht abgedeckt" herleiten. Auch zeigen die Fallstudien des Chaos Reports[Chaos94a], dass der Einbezug der Benutzer das Wichtigste Erfolgskritierum ist.

4.2.2 Grundsatz 2: Einfach und klein

Sowohl die Benutzerschnittstelle als auch der Programmcode sollen einfach sein. Nur konsequentes Refactoring und ständige Kontrolle auf Code Duplikation können das garantieren. Entwicklerteams sollen nie mehr als zwei bis drei Personen umfassen, damit die nötige Kommunikation auf ein Minimum beschränkt wird. Grössere Projekte sollen in Teilprojekte aufgeteilt werden.

Motivation
Wenn ein System einfach und klein gehalten werden kann, ist es sicher einfacher wartbar und anpassbar. Termine und Kosten werden nicht zwingend besser eingehalten, wenn anstelle von wenigen Grossprojekten mehrere kleinere Projekte geführt werden, aber sie sind besser kontrollierbar.

4.2.3 Grundsatz 3: Qualität in allen Teilen

Jeder Teil in einem Projekt soll in der jeweils höchst möglichen Qualität realisiert werden. Dazu gehört eine sorgfältige Planung am Anfang, sauberer Programmcode (auch für Testprogramme und Prototypen), sowie die Vollständigkeit von allen Teilen beim Projektende. Grundvoraussetzung für hohe Qualität ist ein breit abgestütztes Basiswissen von allen Projektmitarbeitern.

Motivation
Dieser Grundsatz ergibt sich direkt aus dem Problem "fehlerhafte Software" und, mittel- und langfristig gesehen, auch aus dem Problem "schlechte Wartbarkeit und Anpassbarkeit".

4.2.4 Grundsatz 4: Zeitplan und Ziele

Der zeitliche Ablauf eines Projektes soll klar geplant sein. Dies umfasst eine Gesamtplanung von Start bis Ende, als auch eine Planung für die einzelne Woche und jeden Arbeitstag. Für jede Zeiteinheit muss ein klares Ziel festgelegt werden. Die Einhaltung der Ziele muss laufend kontrolliert werden. Die Kontrolle führt immer zu Anpassungen am Zeitplan und/oder den Zielen.

Motivation
Der Grundsatz ist eine direkte Reaktion auf das Problem, dass Termine nicht eingehalten werden.

4.2.5 Grundsatz 5: Spass der Beteiligten

Softwareentwicklung ist eine kreative Tätigkeit. Bei solchen Arbeiten ist die Produktivität stark davon abhängig, ob die Beteiligten mit Freude und Begeisterung mitmachen. Voraussetzungen dafür sind ein angenehmes Arbeitsklima (räumlich / personell) und eine persönliche Identifikation aller Beteiligten mit der gestellten Aufgabe.

Motivation
Der Ursprung für diesen Grundsatz ist nicht direkt eines der bekannten Probleme. Es ist vielmehr die Reaktion auf Tatsachen, wie sie in der Einleitung (siehe Kapitel 1) bereits erwähnt wurden. Softwareentwicklung lässt sich nicht nur mit objektiven naturwissenschaftlichen Hilfsmitteln beschreiben. Sie ist mit kreativer Arbeit verbunden und verlangt persönliches Engagement der beteiligten Personen.

5 Fallstudien

5.1 Allgemein

Es gibt verschiedene Gründe dafür, dass Software Engineering nicht im herkömmlichen Sinn, nach naturwissenschaftlichen Kriterien erforscht werden kann. Der Hauptgrund dafür liegt wohl darin, dass es sich bei Softwareentwicklung um eine menschliche Tätigkeit handelt.

- Weinberg umschreibt das im Kapitel "Wie können wir das Programmieren erforschen?" seines Buches[Wein71a] so: *"Da auch das Programmieren eine menschliche Verhaltensweise darstellt, scheint es angebracht, bei der Wissenschaft der Verhaltungsforschung nach zwei Informationen Ausschau zu halten: Ergebnisse der Verhaltensforschung, die wir direkt auf die Softwareentwicklung anwenden können, und Methoden, um die Informationen zu erhalten, die uns nicht direkt zur Verfügung stehen."*
- In einem Paper über Softwareentwicklung[Cock99a], zieht Cockburn folgende Schlüsse aus seinen Beobachtungen: *"Es schein mir, Softwareentwicklung geschieht in der Industrie und nicht an den Universitäten. Universitäten eignen sich hervorragend für Probleme, welche man löst indem man monatelang alleine nachdenkt und experimentiert. Universitäten brachten uns die Automatentheory, komplexe Analysis, Compilers und solche Dinge. Aber Universitäten sind nicht geeignet um zu verstehen, was während der Softwareentwicklung geschieht. "*

Ich teile die Ansicht, dass es zwingend notwendig ist, sich mit der "Realität" zu beschäftigen, um ein vertieftes Verständnis für die Probleme bei der Softwareentwicklung zu erhalten.

Es gibt verschiedene Möglichkeiten, Informationen zu gewinnen. Die Möglichkeiten sind in den folgenden Abschnitten aufgelistet und kurz beschrieben.

5.1.1 Selbstbeobachtung

Die Selbstbeobachtung als Methode um Informationen zu gewinnen wird häufig grundsätzlich abgelehnt. Es ist durchaus so, dass diese Methode ihre Gefahren hat. Es wäre sicher verfehlt, eigene Erfahrungen als allgemein gültige Grundsätze anzusehen. In meinem Fall dienen die eigenen Erfahrungen als Ausgangslage für die fünf formulierten Grundsätze. Die Erfahrung setzt sich zusammen aus der praktischen Arbeit an verschiedenen Projekten der letzten Jahre sowie theoretischem Wissen aus zahlreichen Publikationen, die ich zum Thema Softwareentwicklung gelesen habe.

Um einen ersten Überblick zum Thema zu gewinnen ist die eigene Erfahrung sicher eine gute Ausgangslage. Aber es reicht nicht. Fast wichtiger als die Grundsätze sind deren Grenzen. Die Grenzen lassen sich nur aufgrund einer grossen Zahl von untersuchten Fällen finden. Daraus ergibt sich die Schlussfolgerung, dass ohne Selbstbeobachtung eine derartige Arbeit nutzlos wäre, andererseits ohne breiter angelegte Untersuchung die Selbstbeobachtung allein von fragwürdigem Wert bleibt.

5.1.2 Befragung anderer Personen

Bei der Befragung anderer Personen, wird die Selbstbeobachtung auf eine Menge von Selbstbeobachtungen erweitert. So erhalten wir sicher ein allgemein gültigeres Resultat. Auch bei diesem Verfahren gilt es aber ein paar Punkte zu berücksichtigen:

- Eine Beschreibung ihres Verhaltens und das effektive Verhalten kann sich bei den befragten Personen durchaus unterscheiden. Der Mensch neigt dazu, Dinge so zu sehen und zu beschreiben, wie er es gerne haben möchte und nicht so, wie sie effektiv sind. Eine weitere Quelle für Ungenauigkeiten ist die Tatsache, dass Leute bei Befragungen manchmal auch Antworten geben, die der Befragende gerne hören möchte statt Antworten, die ihrer persönlichen Meinung entsprechen.
- Die Auswahl der befragten Personen kann das Resultat stark beeinflussen. Es ist nicht immer ganz einfach, eine repräsentative Gruppe von Leuten zu finden. Die Leute müssen sich Zeit nehmen und sie geben bei einer Befragung auch immer persönliche Informationen preis. Aus diesen Gründen sind Leute häufiger bereit zu einem Thema ein Interview zu geben, zu welchem sie eine positive Grundhaltung haben.

5.1.3 Experiment

Experimente haben den Vorteil, dass man die Versuchanordnung sehr genau definieren kann und man so ganz gezielte Resultate erhält. Neben den üblichen Problemen, die bei Experimenten mit Personen auftreten (wie zum Beispiel zu eng vordefinierte Versuchsanordnung), gibt es bei Experimenten zum Thema Programmieren und Software Engineering ein paar ganz spezifische Probleme. Weinberg hat das in seinem Buch[Wein71a] ziemlich detailliert beschrieben. Hier nur eine Zusammenfassung der wichtigsten Gründe für die Probleme:

- Bei den meisten Studien, die durchgeführt werden muss der Anteil der Kosten für die Versuchpersonen relativ gering sein. Softwareentwickler sind immer noch sehr gesuchte Leute und so dürfte es schwierig sein, Leute für mehrere Tage oder Wochen gegen ein besseres Trinkgeld für ein Experiment gewinnen zu können. Deshalb werden die Experimente teuer.
- Die Voraussetzungen der Versuchpersonen sind schwer zu erfassen. Messwerte wie "Jahre Programmiererfahrung" oder "geschriebene Zeilen Code pro Tag" sind sicher ungeeignet, um Entwickler zu klassieren. Weinberg verweist auf eine Studie von Sackmann, bei welcher die beiden Programmierer mit der längsten Erfahrung (elf Jahre), einerseits am Besten, auf der anderen Seite aber auch am Schlechtesten abgeschnitten haben.
- Programmieren ist eine individuelle Tätigkeit, also kann man durch Beobachten von Einzelpersonen Resultate erhalten. Softwareprojekte sind aber meistens die Leistung einer Gruppe und die Interaktion zwischen den Gruppenmitgliedern hat erheblichen Einfluss auf die Qualität der Resultate und die Effizienz bei der Arbeit. So gesehen, müsste man eigentlich Experimente mit ganzen Projektteams und nicht mit Einzelpersonen durchführen.

5.1.4 Folgerungen

Experimente sind sehr aufwendig und würden als Methode den Rahmen dieser Arbeit sprengen. Zudem ist der allfällige Aufwand gar nicht gerechtfertig, da die Resultate aus einem Experiment nur beschränkt der Realität entsprechen.
Grundsätzlich sinnvoll erscheint das Durchführen von Befragungen bei Personen, die in Softwareprojekte involviert sind. Diese Resultate, zusammen mit persönlichen Erfahrungen, sollten einen allgemein gültigen Überblick ergeben.

5.2 Befragungen 1. Runde

5.2.1 Inhalt und Ziel

Es gibt viele Möglichkeiten eine Befragung durchzuführen. In einer ersten Runde habe ich die Befragung auf ein halbes Dutzend Personen beschränkt, die ich alle gut kenne. Die Schlüsse aus dieser ersten Befragung führten zu den Anpassungen der Befragung in der zweiten Runde. Bei der Gestaltung des Fragebogens stellten sich verschiedene Probleme:

- Qualitative oder quantitave Beurteilung? Sollen vor allem Fragen gestellt werden, bei welchen der Befragte einfach eine Wertung (z.B. 1-5) abgibt oder sollen Fragen gestellt werden, die als Antwort eine ausführliche Begründung erwarten. Erstere sind bei einer grossen Zahl von Befragten sicher geeigneter. Zweitere sind in der Auswertung aufwendiger, ergeben aber ein besser differenziertes Resultat.
- Offene oder geschlossene Fragen? Soll der Befragte frei antworten können oder soll ihm eine Auswahl von möglichen Antworten angeboten werden?

Im Fragebogen habe ich für jeden der Grundsätze einen Abschnitt erstellt. Jeder Abschnitt enthielt etwa gleich viele Fragen, bei welchen eine Bewertung abgegeben wird (quantitativ) und Fragen, bei welchen eine ausführliche Antwort nötig ist (qualitativ). Bei den Bewertungen waren bis auf wenige Ausnahmen die möglichen Antworten vorgegeben.

Das Ziel war, mit Hilfe der Bewertungen die Grundsätze in ihrer Richtigkeit zu überprüfen, zum Beispiel, wenn für die Arbeit mit einer Software wenig Schulung nötig ist, so kann daraus geschlossen werden, dass der Grundsatz "Einfach" eingehalten wurde. Die Fragen, bei welchen eine ausführliche Antwort erwartet wurde sollten Hinweise geben, wo die Grundsätze möglicherweise ungeeignet oder falsch sind.

5.2.2 Resultate

In der ersten Runde haben sechs Personen einen Fragebogen ausgefüllt, davon drei zum gleichen Projekt. Ausser in einem Fall handelte es sich um kleinere Projekte, die in Zeitspannen von ein bis sechs Monaten abgewickelt wurden. Beim mehrjährigen Projekt handelt es sich um eine Internet Applikation, welche laufend den veränderten Bedürfnissen angepasst werden muss, somit nicht unbedingt ein mittleres Projekt bezüglich Umfang aber sicher bezüglich der Projektdauer und der Anzahl beteiligten Personen.

Alle Befragten beurteilten die Grundsätze mehrheitlich als wichtig. Vorbehalte gab es am ehesten bei "Zeitplan und Ziele". Auch die Frage, ob die Grundsätze eingehalten wurden wird überwiegend bejaht. Die meisten Einschränkungen gab es hier bei "Qualität in allen Teilen".

Weitere Punkte sind bei der detaillierten Durchsicht der Antworten aufgefallen:

- Die Qualität, wie ein Fragebogen ausgefüllt wird ist sehr unterschiedlich. Es ist sehr davon abhängig, ob die Leute gewillt sind, sich Zeit zu nehmen und ob sie in der Lage sind, sich selber zu beobachten und zu beurteilen.
- Quantitative Fragen sind nicht sehr aussagekräftig ausser jene die dazu dienen, das Projekt bezüglich Grösse/Komplexität einzuordnen.
- Allfällige Hinweise oder Erläuterungen der Befragten unter "Bemerkungen" sind am aufschlussreichsten. Allerdings sind auch hier konkrete Fragen nötig. Die Absolutheit der Grundsätze provozierte häufig zu detaillierten Bemerkungen.
- Bei Fragen, in welchen ein Bereich zur Auswahl steht, werden die Extremwerte selten angekreuzt.
- Interessante Schlüsse ergeben sich dort, wo mehrere Sichten auf das gleiche Projekt zur Verfügung stehen.
- Die Grundsätze wurden nirgends völlig in Frage gestellt, was natürlich verschiedene Gründe haben kann:
  - Die Grundsätze werden wirklich als richtig angeschaut.
  - Die Grundsätze sind zu allgemein und deshalb immer richtig.
  - Die befragten Personen stehen mir zu nahe und vertreten zu stark die gleiche Meinungen wie ich.

5.2.3 Schlüsse

Folgende Schlüsse ziehe ich aus den Resultaten der ersten Befragungen:

- Die Antworten auf Fragen, in welchen eine quantitative Bewertung gemacht werden musste, sind nicht sehr aussagekräftig. Um hier gute Resultate zu erhalten, müsste die Befragung vermutlich bei einer grossen Zahl von Personen (hundert oder mehr) durchgeführt werden.
- Direkte Kausalzusammenhänge (im Sinne von wenn - dann Beziehungen) zwischen den gestellten Fragen und den Grundsätzen sind nur schwer abzuleiten. Dies geht aus verschiedenen Bemerkungen, welche die Befragten eingefügt haben, hervor.
- Die Fragen müssen einfach und verständlich sowie möglichst konkret sein. Sonst werden sie gar nicht beantwortet.
- Wenn mehrere Personen zum gleichen Projekt befragt werden, erhält man teilweise widersprüchliche Antworten. Betrachtet man diese Antworten aber im Zusammenhang mit der Funktion der Personen machen sie durchaus wieder Sinn und ergeben ein vollständigeres Bild der Realität.

5.3 Befragungen 2. Runde

5.3.1 Anpassungen Fragen

Aufgrund der Resultate der ersten Befragungen wurden die Fragen wie folgt angepasst:

- Die meisten Fragen, in welchen eine Wertung nötig ist, wurden gestrichen. Es bleiben einzig die Wertungen zu jedem Grundsatz, ob dieser erfüllt wurde und ob dieser als wichtig erachtet wird, sowie die Fragen zur Beurteilung des Erfolges des Projektes.
- Die Skala bei Wertungen wurde auf 1-7 festgelegt. Bei den Auswertungen werden 1 und 2 sowie 6 und 7 zusammengefasst.
- Die Fragen zur Beurteilung des Projekterfolges werden am Anfang gestellt. Neben einer Wertung nach den Kritieren "Bekannte Probleme" (siehe auch Kapitel 2) werden zusätzliche Fragen nach dem Nutzen der Software gestellt.
- Eine neue Fragegruppe zum Thema Vorgehensweise und Methodik wurde eingefügt.
- Neue Fragen, welche vermehrt auf qualitative Beurteilungen und ausführliche Begründungen abzielen, wurden eingebaut.
- Zum Grundsatz "Spass der Beteiligten" wurden persönlichere Fragen hinzugefügt.

Zudem werden die meisten Befragungen in Form eines Interviews durchgeführt. Das hat den Vorteil, dass der Befragte seine Aussagen nicht schriftlich formulieren muss und dass eine bessere Kontrolle über bereits durchgeführte und noch offene Befragungen möglich ist.

5.3.2 Resultate allgemein

In der zweiten Runde wurden insgesamt siebzehn Personen von sechs verschiedenen Projekten befragt. Ausser mit drei Personen wurden die Fragen mündlich beantwortet und nachträglich von einer Tonbandaufnahme niedergeschrieben. Die detaillierten Antworten dieser Befragungen sind im Anhang B aufgeführt.

Im Folgenden die wichtigsten Eckdaten der untersuchten Projekte:

- Zwei Drittel der befragten Personen waren Entwickler, ein Drittel waren Projektleiter oder zuständig für andere Aufgaben (z.B. Support/Dokumentationen).
- Die Projektgrössen variieren von mehreren zehntausend bis mehrere hunderttausend Zeilen Sourcecode.
- Die Projektdauer variiert zwischen einem und fünf Jahren.
- Der investierte Aufwand in Arbeitstagen liegt zwischen vierhundert und über fünftausend Tagen. Ein direktes Verhältnis Grösse/Aufwand ist nicht offensichtlich. Bei nur sechs Projekten ist es gefährlich, daraus schon Schlüsse zu ziehen, aber es zeichnet sich klar ab, dass der Umfang nicht das alleinige Kriterium für den Aufwand ist.
- Verwendete Programmiersprachen waren Java, Javascript, PL/SQL, Shell Scripts, Object Pascal und Cobol.

5.3.3 Resultate Projekterfolg

Die folgende Tabelle gibt einen Überblick über die Bewertungen bezüglich Erfolg des Projektes. Die Skala geht von 1 (sehr schlecht) bis 7 (sehr gut). Der angegebene Wert ist jeweils der Mittelwert der Antworten. In Klammern stehen Minimal- und Maximalwert der Antworten. Die letzte Spalte ist der Mittelwert aller sechs Projekte:

Kritierien	Projekt 1	Projekt 2	Projekt 3	Projekt 4	Projekt 5	Projekt 6	Schnitt
Termine	5	5½ (4-6)	5½ (4-6)	5 (4-6)	2 (2-3)	6	**4**
Kosten	6	6 (5-7)	6 (5-7)	6 (5-6)	4½ (2-6)	-	**5½**
Fehlerfreiheit	6	5 (3-6)	5 (4-5)	5 (3-6)	6	5	**5**
Anforderungen	6	6 (6-7)	5 (4-6)	6 (6-7)	6½ (6-7)	6	**6**
Wartbarkeit	4	5 (4-6)	3½ (2-4)	6	5½ (5-6)	4	**4**

Es ist schwierig, wegen der geringen Projektzahl eine klare Aussage zu machen. Einige Tendenzen sind aber ersichtlich:

- Die Benutzeranforderungen scheinen in allen Projekten gut bis sehr gut erfüllt.
- Bei der Fehlerfreiheit und der Wartbarkeit gibt es die grössten Differenzen bei den Beurteilungen innerhalb eines Projektes.

5.3.4 Resultate verwendete Methode

Bei mittleren und grösseren Firmen kommt Hermes[Hermes], meistens in einer angepassten Form zum Einsatz. Aus verschiedenen Antworten auf andere Fragen lässt sich aber auch bei diesen Projekten schliessen, dass das Vorgehen nicht streng nach dem Phasenmodell strukturiert war. Wie bei den anderen Projekten war es mehr iterativ und gesteuert durch ein "management by doing". Eine klar definierte und nachvollziehbare Methode wurde nirgends verwendet.

Nirgends wurden in irgend einer Form Case-Tools oder andere Programme zur Unterstützung beim Design oder Prozesskontrolle eingesetzt. Einzig beim Entwurf von Datenbanken kamen DB Design Tools zum Einsatz. Sonst begnügte man sich überall mit Word und Visio.

5.3.5 Weitere allgemeine Feststellungen und Beobachtungen

Nicht alle der folgenden Feststellungen sind rekonstruierbar, wenn man einfach die Transkripts der einzelnen Interviews liest. Sie basieren auf Beobachtungen, die während der Interviewdurchführung gemacht wurden:

- Wie ein Problem wahrgenommen und gewichtet wird ist von Person zu Person unterschiedlich. Diese Beobachtung zeigt sich auch bei der Beurteilung der einzelnen Grundsätze (siehe Abschnitt 5.3.6). Zusätzlich zur subjektiven Wahrnehmumg ist vermutlich auch ein unterschiedlicher Informationsstand der einzelnen Personen für diese Differenzen verantwortlich.
- Ich bin nicht ganz überzeugt, dass die Fragen in jedem Fall korrekt und vollständig beantwortet wurden. Dies ist insofern verständlich, als dass aus Selbstschutz und aus Loyalität gegenüber der eigenen Firma niemand gerne über Fehler und Misserfolge spricht.

Interessant scheint mir in diesem Zusammenhang die übereinstimmende Aussage im Projekt 5, dass die Termine schlecht eingehalten wurden. Für mich deutet das darauf hin, dass der Misserfolg in diesem Punkt offen diskutiert und vom Management als begründet akzeptiert wurde. Nur so können die Mitarbeiter dazu stehen, ohne dass damit ihre Selbstachtung darunter leidet.

- Ich habe festgestellt, dass allein schon die Frage zu einem bestimmten Thema die Antworten verändern kann. Aufgrund der Grundsätze oder der Fragen machen sich die Leute Gedanken zum Thema und geben dann eine Antwort, die näher bei einem Wunschdenken, als bei der rückblickenden Beurteilung der Realität liegt.

5.3.6 Übersicht Anwendung und Beurteilung der Grundsätze

Die folgenden Tabellen geben einen Überblick, in welchem Mass die vorgeschlagenen Grundsätze aus Sicht der Projektmitarbeiter umgesetzt wurden und wie weit diese als wichtig beurteilt werden. Weitere Aussagen zu den einzelnen Grundsätzen sind dann im Kapitel 6 im jeweiligen Detailabschnitt integriert.

Wurde der Grundsatz eingehalten?

Grundsatz	Prj. 1	Prj. 2	Prj. 3	Prj. 4	Prj. 5	Prj. 6	Schnitt
Einbezug Benutzer	4	6 (6-7)	4 (1-6)	7	5 (4-6)	6	**5½**
Einfach und klein	4	4 (2-6)	6 (5-7)	5 (4-6)	5 (4-7)	5½ (4-7)	**5**
Qualität	6	5 (3-6)	4 (3-5)	5½ (5-6)	5 (3-6)	4½ (4-5)	**5**
Zeitplan und Ziele	4	5 (3-6)	5 (4-6)	5 (2-6)	2	6	**4½**
Spass	5	4½ (3-6)	5½ (5-6)	6 (6-7)	4½ (2-6)	6 (5-7)	**5½**

Finden Sie den Grundsatz wichtig für den Erfolg eines Projektes? Diese Tabelle ist nicht mehr nach Projekten unterteilt sondern eine Zusammenfassung aller Bewertungen aller befragten Personen der zweiten Runde. Im Titel ist die Bewertung aufgeführt, in der Tabelle die Anzahl Antworten:

Grundsatz	1	2	3	4	5	6	7	Schnitt
Einbezug Benutzer				1		2	14	**7**
Einfach und klein				2	5	6	4	**5½**
Qualität					1	5	11	**6½**
Zeitplan und Ziele				1	6	6	4	**5½**
Spass					2	5	9	**6½**

Auch hier muss man vorsichtig sein bei der Interpretation dieses Ergebnisses, da es sich bei knapp zwanzig Personen doch noch um einen sehr beschränkten Personenkreis handelt. Tendenzen lassen sich trotzdem erkennen:

- Keiner der Grundsätze wird abgelehnt. Das heisst, dass alle einen mehr oder weniger wesentlichen Aspekt für die erfolgreiche Softwareentwicklung abdecken.
- Der Grundsatz "Einbezug Benutzer" wurde von den Befragten klar am höchsten gewichtet, gefolgt von den Grundsätzen "Qualität" und "Spass". Die beiden anderen Grundsätze kommen erst nachher. Dieses Ergebnis wurde auch durch Aussagen zur Frage bestätigt, bei welchen Aspekten die Prioritäten gesetzt werden, wenn wegen äusseren Einflüssen nicht alles erreicht werden kann.

6 Die Grundsätze im Detail

Im folgenden Kapitel werden die Grundsätze im Detail behandelt.

- In einer Einleitung wird genauer ausgeführt, was mit jedem Grundsatz gemeint ist.
- Ein erster Abschnitt enthält eine Zusammenfassung der wichtigsten Ergebnisse der Befragungen.
- Der zweite Abschnitt stellt den Bezug zu wissenschaftlichen Publikationen und Methoden her. Die hier gemachten Angaben sind sicher nicht vollständig, sondern nur ein Bezug zu den bekanntesten Veröffentlichungen.
- Der dritte Abschnitt enthält konkrete Vorschläge, wie der Grundsatz in der Praxis umgesetzt werden kann. Nicht alle Vorschläge sollen und können bei allen Projekten und Teams gleich gut eingesetzt werden. Eine Abwägung der Vor- und Nachteile ist von Fall zu Fall nötig. Die aufgelisteten Vorschläge zeichnen sich dadurch aus, dass die Vorteile meistens überwiegen.
- Der vierte Abschnitt soll die Grenzen des Grundsatzes aufzeigen. Bereits die Befragung in einem relativ kleinen Kreis von Leuten hat bei allen Punkten Vorbehalte aufgezeigt.
- Der fünfte Abschnitt enthält zum Grundsatz passende Zitate und Sprichwörter. Diese haben meistens mehr anekdotischen und nicht wissenschaftlichen Charakter. Sie eignen sich aber gut als Merksätze und Gedankenstützen zu bestimmten Problemen.

6.1 Einbezug der Benutzer

Der Benutzer muss zu jedem Zeitpunkt vollständig in die Projektabwicklung integriert sein. Spezifikation und Diskussion muss in der Sprache des Benutzers stattfinden. Dazu geeignete Hilfsmittel sind Skizzen von Bildschirmmasken und Reports, einfache Ablaufdiagramme, Checklisten und Prototypen. Es empfiehlt sich für jedes Projekt einen "Key User" zu bestimmen, der einerseits mit der Problemstellung und den Abläufen sehr gut vertraut ist und der auch in der Lage ist Wünsche und Anpassungen verständlich zu formulieren.

Es ist ein seltener Idealfall, wenn ein Entwickler neben seinem Informatikwissen auch ein gutes Fachwissen in den Gebieten des Auftraggebers für Softwareprojekte hat. Meistens fehlt das Verständnis gerade in der wichtigen Phase des Projektbeginns.

Umgekehrt ist es auch selten der Fall, dass Benutzer Kenntnisse in Software Engineering haben. So wertvoll UML-Diagramme und ER-Schemas für die Kommunikation zwischen Informatikern sind, für Diskussionen mit Benutzern sind sie ungeeignet. Eine häufige Reaktion von Benutzern auf für sie unverständliche Dokumente ist, dass sie diese einfach ignorieren. Die Entwickler wiegen sich in falscher Sicherheit und Missverständnisse und Fehler in den Anforderungen kommen erst bei Inbetriebnahme der Applikation zum Vorschein.

Einbezug des Benutzers bedeutet für den Benutzer immer auch Aufwand. Es kann nicht automatisch angenommen werden, dass dieser Aufwand auch wirklich betrieben wird. Deshalb

ist es wichtig auch die zu leistenden Arbeiten von Benutzer/Kundenseite zu planen und nicht dem Zufall zu überlassen.

6.1.1 Ergebnisse der Befragungen

In der Beurteilung wurde diesem Grundsatz durchwegs die höchste Priorität zugewiesen. Die Probleme, die dazu führten, dass teilweise trotzdem davon abgewichen wurde, sind unter anderem im Abschnitt 6.1.4 aufgeführt.

Eine häufig wiederkehrende Aussage, gerade auch von den Technikern, war Folgende: "Ein System ist dann erfolgreich, wenn der Benutzer damit zufrieden ist und nicht wenn es technisch möglichst genial umgesetzt wurde." Es scheint, als hätte sich das Berufsverständnis "Informatiker als Dienstleister" bei einer Mehrheit durchgesetzt. Wenn immer noch Applikationen entstehen, die schlussendlich nicht den Kundenbedürfnissen entsprechen, sind die Fehler dafür anderswo zu suchen. Meine persönliche Ansicht dazu ist, dass eines der Hauptprobleme in diesem Fall der schlechte Informationsaustausch über zuviele Zwischenpersonen und mit falschen Hilfsmitteln ist.

In jedem Projekt wurden die Spezifikationen laufend angepasst. Auch dort, wo nach Hermes[Hermes] vorgegangen wird oder dort, wo eine bestehende Software auf neue Technologien umgeschrieben wurde. Die Gründe für Anpassungen waren immer die gleichen:

- Der Benutzer weiss gar nicht genau was er will, oder kann es nicht verständlich formulieren.
- Die Benutzer ändern ihre Anforderungen.
- Die Anforderungen werden von den Entwicklern falsch interpretiert.
- Es treten technische Probleme auf.

Die Tatsache, dass in einem Informatikprojekt Veränderungen die einzige Konstanten sind, wurde durch die Befragungen zu hundert Prozent bestätigt.

Wie häufig die Benutzer neue Versionen erhielten, war sehr unterschiedlich. Der Grund liegt in der Verschiedenartigkeit der Produkte, von "in house" verwendeter Individualsoftware bis zu Standardsoftware, die über den Handel verkauft wird. Es gibt keine Regel, welches Vorgehen das einzig Richtige ist. Umso wichtiger ist es, diesen Punkt bei der Auftragserteilung zu diskutieren und genau festzulegen (siehe auch Abschnitt 6.4).

In direktem Zusammenhang mit der Häufigkeit von Neuversionen steht das angewandte Abnahmeverfahren. Bei wöchentlichen Updates war dieses wenig strukturiert, bei wenigen Updates wurde immer ein mehr formales Verfahren mit geschriebenen Protokollen und Tests durch aussenstehende QS - Leute eingesetzt.

6.1.2 Grundlagen, Publikationen und bekannte Methoden

Das klassische Wasserfallmodell sieht zwar den Einbezug des Benutzers beim Erfassen der Anforderungen vor. In zwei wesentlichen Punkten geht diese Methode aber von falschen Annahmen aus:

- Der Benutzer ist in der Lage, seine Bedürfnisse verständlich, widerspruchsfrei und umfassend zu formulieren.
- Die Bedürfnisse des Benutzers verändern sich nicht.

Andere Methoden berücksichtigen den Einbezug des Benutzers deutlich besser:

- Das Spiral Model von Boehm[Boe88a] sieht ein iteratives Vorgehen vor, bei welchen in jedem Iterationsschritt sowohl die Entwickler als auch die Benutzer besser mit den zu lösenden Problemen vertraut werden.
- Die OOSE Methode von Jacobson[Jac92a] setzt den Schwerpunkt auf "use cases", also Beschreibungen von Szenarien, welche die Interaktion zwischen Benutzer und Softwaresystem darstellen.

Der Benutzer oder Kunde ist in nahezu jedem Softwareprojekt eine unberechenbare Variable. Deshalb führt sicher jede Art von "Light methods" zu einem besseren Einbezug des Benutzers, weil solche Methoden weniger träge sind und so schneller reagiert werden kann.

Am konsequentesten wird der Grundsatz "Einbezug Benutzer" beim Extreme Programming umgesetzt. Das Planspiel ist ein interaktiver Prozess zwischen Benutzer und Entwickler, und bringt einen effizienten Austausch des Wissens. Die "Story cards" werden in der Sprache des Benutzers formuliert. Das iterative Vorgehen mit kleinen Versionen garantiert schnellen Feedback des Benutzers.

6.1.3 Umsetzung in der Praxis

- Projektwörterbuch: Jedes Fachgebiet hat seine eigenen Begriffe oder für allgemeine Begriffe eine ganz spezifische Bedeutung. Um Missverständnisse zu vermeiden empfiehlt es sich, wichtige Begriffe in einem Projektwörterbuch zu beschreiben. Vor allem für Entwickler, welche keinen direkten Kontakt zum Kunden haben, ist ein solches Dokument eine wichtige Hilfe.
  Schreiben sollte das Projektwörterbuch der Projektleiter; jemand von der Benutzerseite sollte kontrollieren, ob die Begriffe korrekt definiert wurden.
- Prototypen auf Papier erstellen: Prototypen sind das geeignetste Hilfsmittel, um gute User Interfaces zu erhalten. Das Verwenden von Handskizzen auf Papier hat gegenüber einem Prototyp auf dem Bildschirm verschiedene Vorteile:
  - Skizzen sind schnell erstellt und geben dem Benutzer einen genügenden Überblick über eine neue Applikation.
  - Änderungen können sehr schnell, zum Beispiel während einer Besprechung gemacht werden. Eine einfache, aber effiziente Technik ist das Verwenden von verschiedenen Farben. So sind in einer Skizze alle Veränderungen immer sichtbar.
  - Skizzen entsprechen formal besser einem Prototypen als ein Programm auf dem Computer. Damit entfällt eine psychologische Schwelle. Der Benutzer sieht eindeutig,

dass es sich um einen Entwurf handelt und wird seine Meinung direkter einbringen. Dank sofort möglichen Änderungen sieht er auch, dass seine Mithilfe Auswirkungen hat und er von den Entwicklern ernst genommen wird.

- Feedback durch Beobachtung: Der Feedback soll nicht nur mündlich oder schriftlich vom Anwender entgegennommen werden, sondern auch durch Beobachtung der Anwender bei der Arbeit gesammelt werden.
  Jeder Entwickler hat eine Vorstellung, wie eine Applikation angewendet werden soll. Diese unterscheidet sich häufig davon, wie der Benutzer mit einem Programm effektiv arbeitet. Eine direkte Beobachtung des Anwenders bringt häufig interessante und wichtige Hinweise auf das reale Benutzerverhalten.

6.1.4 Mögliche Probleme

- Der eigentliche Endbenutzer ist gar nicht bekannt. Dies ist typischerweise bei Entwicklungen für das Internet der Fall. Ein automatisches User Tracking kann gewisse Rückschlüsse über das Benutzerverhalten geben. Auch ungeplante, spontane Gespräche oder Diskussionen über Konkurrenzprodukte können hier wichtige Hinweise über die Benutzerbedürfnisse geben.
- Der Einsatz eines Key-Users birgt immer auch die Gefahr, dass eine Software dann zu stark auf die spezifischen Bedürfnisse dieser Person zugeschnitten wird. Diese Schwierigkeit kann zum Beispiel durch den Einsatz eines Benutzerteams umgangen werden.
- Falls man als Entwickler-Team eines Projektes vom Kunden explizit die Mitarbeit eines Key-Users fordert, besteht eine gewisse Gefahr, dass der gewählte Benutzer nicht aufgrund seines Wissens, sondern weil er gerade Zeit hat, delegiert wird.

6.1.5 Zitate

- Betroffene Benutzer sollen zu Beteiligten gemacht werden.
  (Projektleiter in Befragungen).

Auszug aus "A computer user's manifesto", die zehn Grundrechte des Computer Anwenders:

- Der Benutzer hat immer Recht. Wenn es bei der Anwendung eines Computersystems Probleme gibt, liegt das Problem beim System und nicht beim Benutzer.
- Der Benutzer hat das Recht, die Grenzen der Möglichkeiten eines Computersystems zu kennen.
- Der Benutzer hat das Recht, hilfreiche und durchdachte Antworten zu erhalten, falls irgendwelche Bedenken betreffend des Computersystems auftauchen.

6.2 Einfach und klein

Sowohl die Benutzerschnittstelle als auch der Programmcode sollen einfach sein. Nur konsequentes Refactoring und ständige Kontrolle auf Code Duplikation können das garantieren. Entwicklerteams sollen nie mehr als zirka ein halbes Dutzend Personen umfassen, damit die nötige Kommunikation auf ein Minimum beschränkt wird. Grössere Projekte sollen wenn möglich in einzelne Teilprojekte mit einfachen und klaren Schnittstellen aufgeteilt werden.

Einfach und klein sind eng verknüpfte Begriffe, denn um Systeme wirklich einfach zu machen müssen sie klein sein.

Nicht alle Systeme können klein gehalten werden. Der Mensch besitzt nur sehr beschränkt die Fähigkeit, mehrere Dinge gleichzeitig zu verarbeiten. Um grosse Systeme in den Griff zu bekommen, brauchen wir deshalb Abstraktionen, mit welchen sich solche Systeme übersichtlich darstellen lassen. Das Erstellen solcher Abstraktionen stellt an den Entwickler völlig andere Ansprüche, als das Ausprogrammieren eines einzelnen Moduls. DeRemer und Kron[Dere76a] haben das bereits vor fünfundzwanzig Jahren postuliert.

Einfach und klein beginnt bereits beim Aufnehmen und Definieren der Benutzeranforderungen. Als Analogie sei hier eine Regel aus der Architektur aufgeführt: "Nicht bauen - umnutzen - umbauen - neu bauen". Umformuliert für die Softwareentwicklung heisst das:

- Bringt die geplante Software wirklich eine Verbesserung für den Kunden oder wird nur etwas mit elektronischen Hilfsmitteln gelöst, dass gerade so gut und effizient von Hand erledigt werden könnte?
- Ist es allenfalls möglich, durch den Einsatz eines bestehenden und eingeführten Systems die Bedürfnisse des Kunden abzudecken?
- Gibt es bereits ein System, das mit wenigen Anpassungen die Bedürfnisse des Kunden abdecken könnte?

Erst wenn diese drei Fragen mit "nein" beantwortet werden können, macht es Sinn, eine Software überhaupt zu entwickeln. Diese Feststellung ist aus der Sicht des Softwareentwickler nicht sehr interessant, da er seine Aufgabe vor allem im Erstellen neuer Software sieht. Umnutzen und Umbauen von Software wird aber in Zukunft sicher immer mehr an Bedeutung gewinnen.

6.2.1 Ergebnisse der Befragungen

Die gesamten Antworten lassen sich am Ehesten etwa so zusammenfassen: "Einfach und klein" ist eine häufig nicht erreichbare Wunschvorstellung.

Niemand ist aktiv oder absichtlich von diesem Grundsatz abgewichen. Das Gesetz, dass die Entropie einer Software immer zunimmt[Hunt99a] wurde in den Interviews indirekt bestätigt. Es gab Gründe, die den Vorgang noch beschleunigten:

- Druck von Seiten des Marketings, unnötige Features aufzunehmen, um im Vergleich zur Konkurrenz bestehen zu können.

- Zu knappe Personalressourcen, so dass einzelne Personen für nicht mehr überschaubare Teile verantwortlich waren.
- Wenn es im Team einen dominanten Entwickler gibt, neigen Systeme dazu, schnell einmal monolithisch zu werden.

Weitere bekannte Gründe, warum Systeme nicht einfach und klein bleiben, die in diesen Befragungen nicht explizit erwähnt wurden:

- Es wird versucht, Systeme zu entwicklen, die sehr allgemein sind und die möglichst auch noch alle eventuellen Probleme zu lösen versuchen.
- Es wird zu früh versucht, eine Applikation zu optimieren (Geschwindigkeit und Grösse), bevor diese überhaupt korrekt und zur Zufriedenheit der Benutzer läuft.
- Die Zeit für Refactoring wurde nie eingeplant. Diese Zeit ist auch für neue Applikationen wichtig. Für Projekte, in welchen bestehende Systeme geändert oder ergänzt werden hat das Refactoring eine zentrale Bedeutung.

Wo nicht bewusst Gegenmassnahmen unternommen werden, sind die Projekte irgendwann nicht mehr "Einfach und klein". Das gilt besonders für Projekte, welche über mehrere Jahre gewachsen sind.

In fast allen Projekten sind Funktionen vorhanden, die selten oder gar nie verwendet werden. Es scheint aber dass dies kein Problem ist, da das Ausmass solcher Funktionen sehr beschränkt ist und die Entwickler sehr genau wissen, wo diese vorhanden sind. Ob dieses Wissen auch genügend dokumentiert ist, lässt sich aus den gestellten Fragen leider nicht ableiten.

Die Arbeiten wurden immer nach Fähigkeiten und Ausbildung der Leute aufgeteilt. Der Grund dafür war schlussendlich auch immer der gleiche: Es fehlte die Zeit, um während der Projektarbeit Leute in ein neues Gebiet einzuarbeiten. Wo eine Einarbeitung unumgänglich war, machte eben jeder das, was er schon am besten konnte. In verschiedenen Ausprägungen kam es dann auch zu Problemen, wenn die Schlüsselpersonen eines Teilprojektes nicht verfügbar waren.

Die Probleme mit der Arbeitsaufteilung nach Fachwissen zeigten sich vor allem in den Projekten, in welchen sowohl neue Technologien als auch Hostapplikationen in COBOL verknüpft wurden. Es sind nicht primär die Technikunterschiede, also andere Hardware oder Programmiersprachen, sondern es ist das völlig andere Vorgehen. Entwickler in neuen Technologien sind sich gewöhnt, flexibel zu reagieren und mit den eingesetzten Werkzeugen ist das auch möglich. Hostentwicklungen scheinen viel träger und empfindlicher zu sein. Der kleinste Fehler kann fatale Auswirkungen haben. So wird viel mehr Wert auf umfangreiche Detaildokumentationen und Tests gelegt, was zur Folge hat, das auch kleinste Änderungen mit viel Aufwand verbunden sind.

6.2.2 Grundlagen, Publikationen und bekannte Methoden

Viele bekannte und publizierte Methoden, gerade rund um objektorientierte Softwareentwicklung, sind sehr umfassend und kompliziert. Bereits das Verstehen einer solchen Methode ist sehr aufwendig, geschweige denn das vollständige Erlernen der verwendeten Notationen.

In neueren "light methods" wurde das erkannt, Prozesse und Dokumentationen wurden deshalb wesentlich vereinfacht:

- Cockburn spricht bei seinen Methoden, der Crystal Family zum Beispiel, von "Shrink-to-fit", also eine Reduktion von Design und Dokumentationen auf ein minimales Mass.
- Bei XP kommt der Grundsatz "einfach und klein" bei mehreren Regeln zum Ausdruck:
  - Einfaches Design: Ein einfaches Design ist immer schneller zu erstellen und für andere Personen einfacher zu verstehen.
  - Keine Funktionalitäten auf Vorrat: Es soll nur das implementiert werden, was gerade gebraucht wird. Quasi auf Vorrat erstellte Programmteile bedeuten unnötiger Aufwand und machen ein System schwerfällig.
  - Planning Game: Im Wechselspiel zwischen Kunde und Entwickler wird für eine nächste Iteration definiert, welche Funktionen entwickelt werden sollen. Dabei soll denjenigen Funktionen der Vorzug gegeben werden, die mit kleinstem Aufwand den grössten Nutzen ergeben.

6.2.3 Umsetzung in der Praxis

- Einfachheit in den Benutzeranforderungen: Die Benutzeranforderungen sollen hinterfragt werden. Häufig lässt sich die Komplexität eines Systems wesentlich reduzieren, wenn man auf ein paar wenige Funktionen und Optionen verzichtet. Beschränkung ist eine schwierige Forderung in einem Gebiet wie der Informatik, wo man sich gewöhnt ist, dass alle bekannten Grenzen ständig wieder gesprengt werden.
- Standards einhalten: Jede Programmiersprache und Entwicklungsumgebung kennt gewisse Standards bei Namensgebung und Problemlösungen. Solche Standards soll man einhalten und nur in sehr gut begründeten Fällen davon abweichen.
- Komponenten und Libraries: Dies ist heute wohl die effizienteste Methode, komplexere Programme einfach und klein zu halten. Allerdings ist es auch hier wichtig, dass man sich beschränkt und nicht versucht, mit Hilfe von zuvielen verschiedenen Libraries zu vermeiden, selber Code zu schreiben.
- Respekt vor Schnittstellen: Schnittstellen bilden das Grundgerüst jedes grösseren Softwaresystems. Das können statische Schnittstellen sein, wie zum Beispiel die Struktur einer gemeinsamen Datenbank oder dynamische Schnittstellen, welche den Datenfluss zwischen Teilprogrammen definieren. Die Änderungen an Projektanforderungen führen immer wieder dazu, dass einmal definierte Schnittstellen nicht mehr ganz perfekt sind. Solange eine Schnittstelle aber alle Bedürfnisse abdecken kann, soll sie respektiert und nicht abgeändert werden.
- Keine Tricks: Pointerarithmetik in Sprachen wie C++ und Pascal sind hervorragend dazu geeignet, ein Problem einfach zu lösen; das heisst im Moment einfach und mit etwas weniger Code, dafür ist die Lösung ein Jahr später nicht mehr nachvollziehbar. Solche Programmierertricks sollten grundsätzlich vermieden werden. Einfach und klein kann in einem solchen Fall auch einmal heissen, ein paar Zeilen Code mehr zu schreiben.

6.2.4 Mögliche Probleme

- Wenn ein Projekt in mehrere Teilprojekte aufgeteilt wird, besteht immer die Gefahr, dass sich die einzelnen Teilprojekte verselbständigen: Funktionalitäten werden mehrfach programmiert, Benutzerinterfaces sehen unterschiedlich aus und folgen einer unterschiedlichen Logik. Ein Gesamtleiter, der sowohl auf Stufe Benutzer als auch auf Stufe Entwicklung den Überblick hat, ist unumgänglich. Wegen den unterschiedlichen Anforderungen empfiehlt es sich, für die beiden Bereiche Benutzerschnittstelle und Systemarchitektur unterschiedliche Gesamtleiter zu bestimmen.
- Es gibt Projekte, welche nicht einfach und klein sind und die sich nicht in unabhängige Teilprojekte aufteilen lassen.

6.2.5 Zitate

- Die Komplexität eines Computer Programms ist indirekt proportional zur Qualifikation der beteiligten Entwickler. (Henk Jan, SUMit)
- Etwas ist nicht dann perfekt, wenn man nichts mehr hinzufügen kann, sondern wenn man nichts mehr entfernen kann. (Antoine de Saint-Exupéry)
- Dinge sollen so einfach wie möglich gemacht werden, aber nicht einfacher. (Albert Einstein)

6.3 Qualität in allen Teilen

In jeder Phase eines Projektes soll versucht werden, ein Optimum an Qualität zu erreichen. Dazu gehört eine sorgfältige Planung am Anfang, sauberer Programmcode (auch für Testprogramme und Prototypen), sowie die Vollständigkeit der erstellten Programme und Dokumente beim Abschluss einer Phase. Grundvoraussetzung für hohe Qualität ist neben den Kenntnissen im eigenen Fachgebiet ein breit abgestütztes Basiswissen von allen Projektmitarbeitern.

Qualität ist ein weitläufiger Begriff. Weinberg behandelt diese Frage im Kapitel "Wann ist ein Programm gut?"[Wein71a] und zieht folgenden wichtigen Schluss: "Bei Qualität oder guter Software handelt es sich nicht um objektiv genormte Begriffe oder Begriffe, auf die man sich universell einigen kann oder können sollte." Die Qualität eines Produktes kann man erst beurteilen, wenn auch die Messkriterien und Ziele definiert wurden. Verallgemeinert lässt sich Qualität etwa so definieren: "Qualität heisst etwas ist für jemanden wertvoll". Im Folgenden bezieht sich Qualität am ehesten auf die Form des Resultates und weniger auf Termine, Kosten oder Verkaufserfolg.

Qualität ist nicht eine Frage des Umfangs. Eine sorgfältige Planung am Anfang umfasst vielleicht nur eine Handskizze der wesentlichen Systemteile, dafür ist sie korrekt und auf der Stufe des gewählten Detaillierungsgrads vollständig.
Das gleiche gilt für Dokumentationen. Qualitativ gute Dokumentationen sind kurz und prägnant und enthalten nur gerade das, was zum Beispiel nicht direkt beim Gebrauch der Software oder beim Lesen des Sourcecodes ersichtlich ist. Zu umfangreiche oder falsche Dokumentationen wirken sich sogar eher schädlich auf die Gesamtqualität eines Projektes aus.

6.3.1 Ergebnisse der Befragungen

Alle Personen beurteilten Qualität als wichtige Anforderung an erfolgreiche Software. Allerdings sind in zwei Richtungen Abweichungen erkennbar:

- Zeitdruck führt dazu, dass die gewünschte Qualität nicht erreicht wird.
- Management und Projektleiter sind eher bereit, zu Gunsten von Einhaltung von Terminen und Kosten bei der Qualität Abstriche vorzunehmen. Sie sehen Qualität immer zusammen mit dem Preis, den sie hat.

Hundertprozentige Fehlerfreiheit gibt es nicht. Das zeigte auch diese Befragung. Bei den Fehlern handelte es sich durchwegs um Details, oder solche, die für den aktuellen Betrieb nicht von Bedeutung sind. Genau aus diesen Gründen werden sie auch nicht behoben.

Tests werden überall durchgeführt, allerdings in unterschiedlicher Form. Automatisierte Tests werden praktisch nirgends eingesetzt. Bei Individualsoftware für nicht kritische Arbeiten finden Tests häufig direkt in der produktiven Umgebung statt. Bei kritischeren Anwendungen oder bei Produkten, welche als Standardsoftware vertrieben werden, wurden vermehrt vordefinierte Testabläufe durchgeführt und die Anwendung auf Korrektheit überprüft. Allerdings wird auch bei diesen strukturierten Tests viel von Hand erledigt.

Ähnlich sieht die Situation beim Design- und Codereview aus. Es gibt zwar bei den meisten Firmen mehr oder weniger ausgearbeitete Richtlinien, aber gerade auf Stufe Implementation gilt vorwiegend das Prinzip "Selbstkontrolle". Als Gründe dafür werden mangelnde Zeit und Ressourcen aufgeführt.

Ausser bei einem Projekt, in welchen nach Hermes[Hermes] vorgegangen wurde, wird die Dokumentation zwischen hinreichend und ungenügend beurteilt. Wieder scheint die fehlende Zeit der Hauptgrund zu sein. Mehrfach wurde aber in diesem Zusammenhang bemerkt, dass im Zweifelsfall die Zeit besser in sauber geschriebenen Code investiert wird, da dieser immer noch die am meisten gültige und verbindliche Dokumentation sei.

Was heisst sauberer Code? Dieser Begriff wurde verschiedentlich in Antworten verwendet, ohne dass er genauer beschrieben wurde. Es scheint aber ein implizites Verständnis unter Softwareentwicklern für diesen Begriff zu geben:

- Einhalten von Formatierungen (Einrücken, Abstände).
- Einhalten von Richtlinien der verwendeten Programmiersprache (Konventionen für die Schreibeweise von Schlüsselwörtern und die Namensvergebung).
- Gute Kommentare (siehe Artikel von Pintér Gábor[Gabor01a]).

6.3.2 Grundlagen, Publikationen und bekannte Methoden

In den Anfängen des Software Engineering lag die Qualitätssicherung alleine in der Verantwortung der einzelnen Programmierer. Standards wurden dann bei militärischer Software eingeführt, welche sich in den siebziger Jahren schnell in die kommerzielle Softwareentwicklung ausbreitete. Bertrand Meyer formulierte Ende der achtziger Jahre folgende Qualitätskriterien[Mey88a]:

- Correctness: In welchem Mass ist die Software fehlerfrei und macht das, was von ihr erwartet wird?
- Robustness: Funktioniert die Software auch unter unvorhergesehenen abnormalen Bedingungen korrekt?
- Extendibility: Wie einfach kann die Software erweitert und an geänderte Benutzeranforderungen angepasst werden?
- Reusability: Kann die Software ganz oder teilweise für andere, neue Applikationen verwendet werden?
- Compatibility: Kann die Software einfach mit anderen Applikationen kombiniert werden, zum Beispiel für den Datenaustausch?

Die wohl bekanntesten Qualitätssicherungswerkzeuge sind die von ISO publizierten Standards. Wichtig für die Softwareentwicklung ist der Standard ISO 9001. Dieser Standard enthält zwanzig Anforderungen, welche für eine effiziente Qualitätssicherung erfüllt sein müssen. ISO 9001 ist generell auf Ingenieur- und Dienstleistungsunternehmen ausgerichtet. Für Softwareprozesse gibt es eine spezielle Richtlinie, die ISO 9000-3. Die ISO Richtlinien zielen primär auf die Qualitätssicherung des Softwareerstellungsprozesses. Sie enthalten keine Angaben über Qualitätskriterien für die Software an sich. Ausführliche Details zu den ISO Richtlinien sind auf zahlreichen Seiten im Internet zu finden.

Als wichtiges Modell, um die Qualität des Softwareerstellungsprozesses in einem Unternehmen zu beschreiben, wurde an der Carnegie Mellon University vom Software Engineering Institute das Capability Maturity Model (CMM) entwickelt. Dieses Modell hat sich heute als de facto Standard für die Beurteilung und Verbesserung von Prozessen beim Entwickeln von Software etabliert. Das CMM klassiert die Prozesse von Unternehmen auf fünf verschiedenen Stufen:

- Initial: Die Prozesse sind ad hoc oder chaotisch organisiert. Der Erfolg ist im wesentlichen von Einzelleistungen abhängig.
- Repeatable: Ein einfaches Projektmanagement für die Terminplanung, Kosten- und Funktionskontrolle wird eingesetzt.
- Defined: Die Prozesse aller Tätigkeiten sind dokumentiert und standardisiert.
- Managed: Es werden detaillierte und quantifizierte Kontrollen der Prozesse und für die Produkte eingesetzt.
- Optimized: Die kontinuierliche Verbesserung der eigenen Prozesse ist integraler Bestandteil der Tätigkeit des Unternehmens.

Um von einer Stufe zur nächsten zu gelangen, müssen klar definierte Voraussetzungen erfüllt sein. Detaillierte Informationen zum CMM sind auf der Internetseite des Software Engineering Institute zu finden

Ein Vergleich von ISO 9001 mit dem CMM zeigt, dass CMM die Anforderungen von ISO 9001 nahezu vollständig abdeckt. Das Gegenteil ist nicht der Fall. ISO 9001 beschreibt eine Minimalmenge von Kriterien für ein ausreichendes Qualitätsmangagementsystem. Im Gegensatz dazu bietet das CMM auch Unterstützung zur Verbesserung der Entwicklungsprozesse.

Rückblickend auf die letzten zwanzig Jahre stellt Pressman[Pres97a] fest, dass die Eigenschaften, welche qualitativ gute Software auszeichnen, eine der wenigen Konstanten der Informatik war. Softwareunternehmen, welche die Kritierien in ihrem Betrieb im Sinne einer Checkliste umsetzen haben deshalb Gewähr, auf längere Sicht Systeme zu entwickeln, welche stabil funktionieren und die Benutzerbedürfnisse immer optimal abdecken.

Neben diesen allgemein gültigen Aussagen gilt für "Light methods", welche eine inkrementelle Entwicklungsstrategie verfolgen, wie zum Beispiel Extreme Programming, dass die Qualitätskriterien laufend auf einfache Weise überprüft werden können. XP trägt dem Rechnung, indem automatisierte Tests ein integraler Bestandteil der Methode sind.

6.3.3 Umsetzung in der Praxis

- Kontrollierbare Abmachungen mit Kunden: Die allgemeinen Qualitätskriterien (Correctness, Maintainability, Usability) müssen bei einem Softwareprojekt konkret und in sowohl für Kunden wie Entwickler verständlicher Form festgehalten werden. Zudem soll es sich um einfach messbare Kritierien handeln wie "Anzahl aufgetretene Fehler im Testbetrieb über X Tage".
- Gute Namensgebung und keine Code Duplikation: Die Erfahrung in verschiedenen Projekten hat gezeigt, dass auf der Stufe Implementation der Verstoss gegen diese beiden Regeln zu den häufigsten Ursachen für schlecht wartbare Programme gehören. Gerade die Verwendung von RAD Tools verhindern zwar nicht, dass die Regeln eingehalten werden können, sie verleiten aber den Entwickler dazu, gegen die Regeln zu verstossen (siehe Artikel von Marco Cantù[Cantu99a]). Es wird sehr aufwendig, diese Fehler im nachhinein zu korrigieren. Auch ist es schwierig gegenüber dem Management oder Auftraggeber den nötigen Aufwand zu begründen, da das einzige, was der Benutzer bei dieser Arbeit mitbekommt, allfällig neu eingebaute Flüchtigkeitsfehler sind.
- Tests: In welcher Form auch immer getestet wird (automatische Tests, definierte manuelle Tests oder unstrukturiertes Testen), die Tests sollten von jemand anderem als dem Entwickler durchgeführt werden. Das entbindet den Entwickler nicht davon, jeweils seine Arbeit zu überprüfen. Aber aus menschlichen Gründen ist im Unterbewussten niemand bereit, etwas, dass er gerade mit viel Energie erstellt hat als fehlerhaft anzuschauen.
- Reflektieren bei Fehlern: Bei jedem Fehler, der gefunden wird, soll man sich bei der Korrektur des Fehlers die folgenden drei Fragen stellen:
  - Kommt dieser Fehler auch an anderen Stellen in der Applikation vor?
  - Verstecken sich hinter dem Fehler weitere Folgefehler?
  - Was kann ich tun, um in Zukunft solche Fehler zu vermeiden?

  Diese Fragen werden im Artikel von Tom Van Vleck[Vleck89a] detailliert behandelt. Die Idee hinter den drei Fragen ist, dass jeder Fehler, der auftritt, nur ein Symptom im darunter liegenden Prozess ist. Nur die Symptome zu behandeln reicht nicht, sonst treten diese immer wieder auf. Es ist nötig herauszufinden, was die genaue Ursache für den Fehler war um sicherzustellen, dass der Fehler mit der Korrektur wirklich behoben wird.

6.3.4 Mögliche Probleme

- Je lückenloser eine Methode zur Qualitätssicherung versucht, alle möglichen Eventualitäten abzudecken, desto mehr verkommt sie zur reinen Kontrolle der im Projekt beteiligten Personen. Tendenziell sinkt damit die Eigenverantwortung, was sich im schlimmsten Fall sogar negativ auf das Resultat auswirken kann.

6.3.5 Zitate

- The problem of quality management is not what people don't know about it. The problem is what they think they do know….
  In this regard, quality has much in common with sex. Everybody is for it. (Under certain conditions, of course.) Everybody feels they understand it. (Even though they wouldn't want to explain it.) Everyone thinks execution is only a matter of following natural inclinations (After all, we do get along somehow.) And, of course, most people feel that problems in these areas are caused by other people. (If only they would take the time to do things right.) (Roger S. Pressman).
- Durch Tests kann nie die Fehlerfreiheit bewiesen werden, sondern nur die Tatsache, dass es Softwarefehler gibt (Roger S. Pressman).

6.4 Zeitplan und Ziele

Der zeitliche Ablauf eines Projektes soll klar geplant sein. Dies umfasst eine Gesamtplanung von Start bis Ende, als auch eine detaillierte Planung, jeweils für die nächsten Phasen. Die Unterteilung einer Detailplanung erfolgt wochenweise bis tagesweise für spezielle Aktionen. Für jede Zeiteinheit muss ein klares Ziel festgelegt werden. Die Einhaltung der Ziele muss laufend kontrolliert werden. Die Kontrolle führt immer zu Anpassungen am Zeitplan und/oder den Zielen.

Brooks stellt im Essay "Hatching a Catastrophe"[Brook75a] fest: überhaupt einen Zeitplan zu haben, ist der erste Schritt, um ein Projekt im zeitlichen Ablauf zu kontrollieren. Der nötige Detaillierungsgrad ist sehr stark von der Art des Projektes abhängig. So braucht es zum Beispiel für ein kleines Programm, das für eine Firma von geringer Bedeutung ist einen viel weniger stark strukturierten Zeitplan, als für ein grosses Projekt mit existenzieller Bedeutung.

6.4.1 Ergebnisse der Befragungen

Aufwandabschätzung und Zeitplanung zeigten sich in den Interviews als etwas vom Schwierigsten in einem Softwareprojekt. Verschiedene Gründe wurden aufgeführt:

- Es fehlt die persönliche Erfahrung von jedem Mitarbeiter, den Aufwand für eine bestimmte Arbeit abschätzen zu können.
- In Softwareprojekten treten immer wieder kleinste Detailprobleme auf, welche einen Zeitplan völlig durcheinander bringen können.

- Der Zeitplan wird sehr stark von aussen beeinflusst, sei es zum Beispiel durch Termindruck von Marketingabteilungen oder dadurch, dass der Auftraggeber seinen Beitrag zum Projekt (Grundlagen bereitstellen, Kontrollen) nicht termingerecht leistet.

Zeitpläne wurden aber überall erstellt, fast immer in schriftlicher Form und sie waren auch allen Mitarbeitern bekannt. Dass eine Planung auf Tage genau gemacht werden soll, wird mehrheitlich als wenig sinnvoll erachtet, ausser für spezielle Aktionen wie zum Beispiel Migrationsprozesse oder bei Entwicklern mit wenig Programmiererfahrung.

Mehrfach wurde auch erwähnt, dass unrealistische, nicht einhaltbare Zeitpläne auf die Dauer immer zu Problemen führen. Die erreichte Qualität nimmt ab und schlussendlich leidet darunter auch das Arbeitsklima und die Zufriedenheit der Mitarbeiter.

6.4.2 Grundlagen, Publikationen und bekannte Methoden

Neben schlechter Wartbarkeit war das Nichteinhalten von Terminen schon früh ein Thema, mit welchem sich Publikationen zum Thema Software Engineering befassten. Spielten am Anfang die Kosten noch nicht so eine grosse Rolle, ist dieser Faktor heute eng mit den Terminen verbunden. Nichteinhalten von Terminen heisst praktisch immer auch nicht Einhalten der Kosten.
Allgemein ist die Zeitplanung einer der zentralen Inhalte des Projektmanagements. Es wurde in zahlreichen Publikationen umfassend behandelt. Die Methoden lassen sich generell etwa wie folgt umschreiben:

- Gesamtprojekt wird in einzelne kleinere und überschaubare Aufgaben unterteilt.
- Jede Aufgabe umfasst ein Anfangs- und Enddatum, den geplanten Aufwand in Arbeitstagen, eine Beschreibung des zu erreichenden Ziels sowie die dafür verantwortlichen Personen.
- Zwischen den einzelnen Aufgaben gibt es Abhängigkeiten, manche können parallel ausgeführt werden, manche müssen hintereinander erfolgen. Aus allen diesen Abhängigkeiten lässt sich der kritische Pfad für das Gesamtprojekt ableiten.
- Meilensteine nach Abschluss von einzelnen Aufgaben oder Gruppen von Aufgaben geben dem Projekt den äusseren Rahmen. Zu jedem Meilenstein gehören Reserven, je nach Risiko und Umfang der zu erledigenden Aufgaben

Bei Softwareprojekten gibt es allerdings wesentliche Punkte, in welchen sich diese von anderen Ingenieurprojekten unterscheiden. Es gibt durchaus viele Parallelen aber es gibt auch wesentliche Unterschiede, welche sich bei gerade bei der Zeitplanung auswirken:

- Bei den meisten Ingenieurprojekten nimmt die Konstruktions- oder Ausführungsphase mehr als fünfzig Prozent der Zeit in Anspruch. Bei einem Softwareprojekt dagegen sind es genau genommen gerade mal der Bruchteil von einem Prozent, nämlich dann, wenn der Compiler gestartet wird. Die ganze übrige Zeit wird von allen Personen bis und mit jedem Entwickler Entwurfsarbeit geleistet.
- In fast allen Softwareprojekten ist man in irgend einer Form mit neuer Technologie konfrontiert. Es fehlt das Know-How, gute Aufwandabschätzungen machen zu können und die Technologie effizient anzuwenden.

- Quantitative Angaben, wieweit eine Aufgabe fertig ist, sind wenig aussagekräftig. "90% des Sourcecodes ist etwa nach der halben Zeit geschrieben, die Fehlersuche ist praktisch immer zu 99% abgeschlossen. Das einzige was zählt, sind zu 100% abgeschlossene Aufgaben."[Brook75a]

Das Problem der Aufwandabschätzung versucht Humphrey in seinem Personal Software Process (PSP)[Hum95a] zu lösen. Er beschreibt, wie jeder Entwickler für sich selber seine Effizenz messen kann und mit den so gefunden Zahlen den Aufwand für zukünftige Projekte besser abschätzen kann. Darauf aufbauend definiert Humphrey den Team Software Process (TSP), welcher Entwicklerteams bei der Prozessplanung und Aufwandabschätzung unterstützen soll.

6.4.3 Umsetzung in der Praxis

- Entwickler einbeziehen in Zeitplanung: Aufwandabschätzungen sind etwas vom Schwierigsten bei der Softwareentwicklung. Deshalb sollten die realisierenden Personen unbedingt einbezogen werden. Im allgemeinen führt das zu folgenden Vorteilen:
  - Die Zeitpläne sind meistens realistischer. Unerfahrene Entwickler sind zwar häufig grosse Optimisten, aber gerade für sie ist es auch wichtig, sich mit dieser Frage auseinanderzusetzen und so überhaupt Erfahrungen machen zu können.
  - Weil sie mitreden konnten, sind die Entwickler automatisch mitverantwortlich für die Zeitpläne und deshalb auch eher bemüht, diese einzuhalten.
  - Eine Auseinandersetzung mit dieser wichtigen aber fast immer unangenehmen äusseren Randbedingung vermittelt den Entwicklern eine bessere Gesamtsicht auf das ganze Projekt.
- Eine Arbeitswoche = dreissig Stunden: Die meisten Betriebe in der Schweiz haben ungefähr vierzig Stunden offizielle Arbeitszeit pro Woche, dazu kommen, je nach Firma mehr oder weniger Überstunden. Es ist aber eine Tatsache (gilt nicht nur für Informatikprojekte), dass im Normalfall maximal dreissig Stunden für die effektive Arbeit an Projekten zur Verfügung steht. Die restliche Zeit wird für wichtigere und unwichtigere Alltagsprobleme verwendet wie Papierstau in Drucker oder Kopierer beheben, Stunden- und Spesenrapporte schreiben, neue Software installieren, Newsgroups und Websites lesen, Emails beantworten, Kaffee/Apéro mit wichtigem Kunden, nicht zu vergessen die kleinen "Zwischendurch-Aufträge" und schlussendlich noch Ferien und Krankheit.
- Änderungen einplanen: Jedes Informatikprojekt ist neu und hat seine Risiken, seien diese technischer Art oder in Form von geänderten Benutzeranforderungen. Es lohnt sich deshalb von Anfang an, Zeitreserven in der Grössenordnung von bis zu 50% vorzusehen und diese so geschickt zu verteilen, dass nicht das Projekt erst begonnen wird, wenn diese Reservezeit schon mal aufgebraucht ist. Und falls ein Projekt dann ausnahmsweise mal zu früh abgeschlossen wird, sollte man deshalb nicht für weitere Projekte wieder engere Zeitpläne aufstellen.
- Kommunikation kostet Zeit: Brooks[Brook75a] hat diesen Effekt sehr klar beschrieben. Schon in kleinen Teams spielt der Informationsaustausch eine wichtige Rolle. Neben der Zeit die es braucht, um eine Mitteilung zu machen, braucht der Empfänger auch Zeit, die Information aufzunehmen, sie zu verarbeiten und darauf zu reagieren.

6.4.4 Mögliche Probleme

- Eine zu detaillierte zeitliche Planung führt schnell zu grossem Aufwand, den Plan immer aktuell zu halten. Die Gefahr ist dann gross, dass der Zeitplan überhaupt nicht mehr nachgeführt wird.
- Für Projektleiter ist der Druck von aussen, durch Kunden oder zum Beispiel Marketingabteilungen, häufig sehr gross, so dass sie Zeitplänen zustimmen sollen oder müssen, von denen sie genau wissen, dass sie unrealistisch sind.
- Es ist vor allem dann schwierig auf Abweichungen im Zeitplan zu reagieren, wenn diese nicht wegen eines grossen Problems auftreten, sondern aus vielen einzelnen Gründen, welche erst in ihrer Totalität ein Problem ergeben.

6.4.5 Zitate

- Adding manpower to a late software project makes it later. (F. Brooks)
- How does a project get to be a year late? - One day at a time. (F. Brooks)
- Personen unter Zeitdruck arbeiten nicht besser, sie arbeiten nur schneller. (DeMarco/Lister)

6.5 Spass der Beteiligten

Softwareentwicklung ist eine kreative Tätigkeit. Bei solchen Arbeiten ist die Produktivität stark davon abhängig, ob die Beteiligten mit Freude und Begeisterung mitmachen. Voraussetzungen dafür sind ein angenehmes Arbeitsklima (räumlich / personell), funktionierende Kommunikation unter den Projektmitgliedern und eine persönliche Identifikation aller Beteiligten mit der gestellten Aufgabe.

Das Arbeitsklima, in welchem Entwickler arbeiten, ist vielschichtig und schwer durchschaubar. Es ist voll von zwischenmenschlichen Beziehungen und dem Einfluss einzelner Persönlichkeiten. Im Grundsatz "Spass der Beteiligten" steckt vor allem die Erfahrung, dass die entscheidenden Aspekte in der Softwareentwicklung menschliche sind und nicht technische. Umgesetzt auf den Erfolg oder Misserfolg von Projekten heisst das, dass das Resultat mehr davon abhängt, wer die Arbeit ausgeführt hat und nicht wie die Arbeit ausgeführt wurde.

Der "Human Factor" wird fast immer unterschätzt. Softwareentwicklung ist eine kreative Tätigkeit, eine Entwurfsarbeit, die vor allem im Kopf aber auch bei der Kommunikation von mehreren Personen stattfindet. Deshalb erfordert sie, gegenüber einem herkömmlichen Produktionsprozess aus der Industrie, völlig andere Massnahmen.

6.5.1 Ergebnisse der Befragungen

Die wichtigste Motivation bei den befragten Personen, war immer die technische Herausforderung oder die Grösse des Projektes. Persönlich fand ich es erstaunlich, dass "Geld verdienen" nie erwähnt wurde, auch nicht nebenbei als Randbemerkung.

Als negativ wurde am häufigsten der zu starke Zeitdruck angeführt. Vereinzelt wurden auch die Probleme mit externen Stellen und zwischenmenschliche Probleme im Projektteam erwähnt. Mehrheitlich wurde aber gerade die Zusammenarbeit im Team als besonders positive Erfahrung empfunden.

Konflikte zwischen Personen sind auch überall aufgetreten, allerdings in sehr unterschiedlicher Art und Ausmass. Kleinere Probleme konnten durch Gespräche bereinigt werden. Grössere Konflikte hatten immer in irgendeiner Form personelle Konsequenzen. Teams wurden umstrukturiert, einzelne Mitarbeiter ausgetauscht (nur bei grösseren Firmen) oder einzelne Mitarbeiter haben gekündigt. Die Projektleiter spielten bei der Konfliktlösung immer eine zentrale Rolle. Sie übernahmen die Gespräche und Vermittlungsarbeit. Andere externe, neutrale Personen wurden nie beigezogen.

Gesellschaftliche Anlässe finden in sehr unterschiedlichem Ausmass statt. In Firmen / Projekten, wo es solche Anlässe gibt, nimmt jedermann teil. In Firmen, wo solche Anlässe fehlen, wird das von allen befragten Personen bedauert.

Spass wird als wichtige Voraussetzung für erfolgreiche Projekte angeschaut, allerdings mit ein paar konkreten Vorbehalten:

- Die letzten fünf Prozent in einem Projekt sind fast immer Fleissarbeit und machen selten Spass. Sie sind aber ebenso unerlässlich für ein erfolgreiches Projekt.
- Spass ist vielleicht der falsche Begriff, da er im Zusammenhang mit Arbeit negativ behaftet ist. Freude oder Motivation sind passendere Wörter.

6.5.2 Grundlagen, Publikationen und bekannte Methoden

The Psychology of Programming[Wein71a], Peopleware[DeMar99a] und die Artikel von Cockburn[Cock99a] sind Publikationen, welche sich alle sehr konsequent mit dieser personenzentrierten Sicht zum Thema Softwareentwicklung befassen.

- Das Buch von Weinberg erschien erstmals vor bald dreissig Jahren. Es war die erste Publikation, welche Softwareentwicklung vom personenzentrierten Standpunkt aus näher untersuchte. Das Buch ist in vier Abschnitte aufgeteilt:
  - Programmieren als menschliche Leistung
  - Programmieren als soziale Tätigkeit
  - Programmieren als individuelle Tätigkeit
  - Programmierwerkzeuge

  Der letzte Abschnitt ist heute naheliegenderweise ziemlich überholt und liest sich mehr als historische Anekdoten, die ersten drei Abschnitte sind aber im Grossen und Ganzen immer noch gültig. Weinberg hat bereits vor dreissig Jahren wichtige Prinzipien von Extreme Programming (XP) mehr oder weniger klar formuliert:

- Egoless programming: Weinbergs Betrachtungen entsprechen mehr oder weniger der Regel "Collective Code Ownership".
- Vier-Augen-Prinzip: Hier wird die wohl bekannteste XP Regel, das "Pair Programming" andeutungsweise bereits formuliert.
- Testen: Auch die Regel "Code the Unit Test First" hat Weinberg bereits beschrieben, um den Programmierer davor zu schützen, das Testen allzu rasch abzubrechen.

- Peopleware ist knapp fünfzehn Jahre alt. Es werden die folgenden Punkte, vor allem mit Erfahrungsbeispielen, detailliert behandelt:
  - Softwareentwickler können nicht als austauschbare Bausteine behandelt und geführt werden.
  - Falsche Arbeitsplatzgestaltung und Büroaufteilungen können die Arbeit wesentlich behindern oder gar verunmöglichen.
  - Der Erfolg von Softwareprojekten beginnt mit der Anstellung der richtigen Leute und der Formung von Teams. Gute Teams können nicht aktiv geformt werden, die Teambildung kann aber durch falsche Massnahmen verhindert werden.
  - Professionelle Arbeit kann und soll auch Spass machen.

- Cockburn schrieb unter anderem verschiedene Artikel zum Thema "Personen in Softwareprojekten" und hat darin verschiedene für die Softwareentwicklung relevanten Charakteristiken von Personen aufgezeigt.[Cock99b]

Interessant ist, das sehr vieles, was in diesen Büchern geschrieben steht, bis heute nichts an Aktualität verloren hat. Die technischen Möglichkeiten haben sich stark weiterentwickelt, aber das Verhalten der beteiligten Personen hat sich über die Jahre nicht gross verändert. Zwei wichtige Erfahrungen, die in allen Publikationen auftauchen:

- Jede Organisation hat neben der formellen auch eine informelle Struktur, welche für den Informationsaustausch zwischen Personen sehr wichtig ist. Während man für erstere fast überall schöne Organigramme hat, sind zweitere nicht offensichtlich und werden manchmal auch unabsichtlich zerstört.

- "Trust in people to do what is necessary" hat es Alistair Cockburn formuliert oder "Kontrolle ist gut, Vertrauen ist besser" DeMarco/Lister in Peopleware. Beide Aussagen umschreiben die Erfahrung, dass jeder Mitarbeiter auch eine mitdenkende Person ist und eigentlich fast immer auch am Erfolg interessiert ist und seine Arbeit entsprechend ausrichtet.

6.5.3 Umsetzung in der Praxis

- Gemeinsame Pausen und gesellschaftliche Anlässe: Menschen sind kommunikative Wesen und wollen auch kleinste Erfolge und Probleme anderen Personen mitteilen. Kaffeepausen sind dazu manchmal besser geeignet als offizielle Projektmeetings. Sie sind Teil der informellen Struktur in einem Betrieb. Gesellschaftliche Anlässe, welche mit der eigentlichen Arbeit wenig zu tun haben geben den Leuten die Möglichkeit, Gemeinsamkeiten neben der Berufsarbeiten kennenzulernen und auszubauen. Als Folge davon können so fachliche Konflikte in Projekten einfacher gelöst werden, weil die Kommunikation auf verschiedenen Ebenen möglich ist. Einfach gesagt, man versöhnt sich wieder bei der übereinstimmenden Beurteilung des Fussballspiels vom Vorabend.

- Ziele kontrollieren und nicht (Präsenz-)zeit: Entwickler lösen ihre Aufgaben auf ganz verschiedene Weise. Es gibt solche, welche mehr oder weniger linear vorwärtskommen, andere haben mehr eine sprunghafte Arbeitsweise und machen zwischendurch scheinbar nichts oder unnötige Dinge. Gerade bei letzteren ist diese leere Zeit auch ein wesentlicher Bestandteil der Arbeit und es wäre falsch zu versuchen, solche Leute dort zu mehr Effizienz zu mahnen. DeMarco/Lister schreiben dazu passend "Visuelle Überwachung von Entwicklern ist sinnlos, Strafgefangene werden visuell überwacht."
- Anerkennung der Leistungen: Menschliche Wesen brauchen von Zeit zu Zeit die Bestätigung, das sie auf dem richtigen Weg sind. Jede einer Person ausgesprochene Anerkennung ist für diese ein Erfolgserlebnis. Gerade auch in einem Team können solche gemeinsamen Erfolgserlebnisse die Teambildung fördern.
- Freiraum für Experimente: Jeder gute Softwareentwickler ist ein Forscher und will in irgendeinem Gebiet, sein Fachwissen ausbauen. Diese Tätigkeit der gezielten oder auch planlosen Weiterbildung sollte in der regulären Arbeitszeit stattfinden könne. Das heisst, es muss die nötige Zeit implizit oder explizit dafür vorgesehen werden.

6.5.4 Mögliche Probleme

- Eine zu starke Verbindung von Privat- und Arbeitsleben kann sich unter Umständen auch negativ auswirken. Zudem wird diese Verbindung auch nicht von allen Personen gewünscht.
- Auch wenn zwischenmenschliche Faktoren den Projekterfolg massgebend beeinflussen, bilden solide fachtechnische Kenntnisse jedes einzelnen Mitarbeiters immer noch eine wichtige Grundlage und dürfen nicht komplett vergessen werden. Wobei anzumerken ist, dass es einfacher ist, fehlende technische Kenntnisse zu erlernen, als nichtfunktionierende zwischenmenschliche Beziehungen nachträglich zu korrigieren.

6.5.5 Zitate

- People are pretty much the same as they ever were. They are diverse, they have failure modes, they have success modes, they are non-linear. They work in certain ways better than others. They need to communicate. Software creation is limited by human expression. (Alistair Cockburn)
- Anonyme Hochhaussiedlungen und überfüllte Züge zum Arbeitsort befriedigen unsere Bedürfnisse bezüglich Gemeinschaft nicht. Teams können das ersetzen, wenn sie funktionieren. Vielleicht sind sie deshalb für uns so wichtig. (DeMarco/Lister)
- Skill may be hard to come by, but it's always easier to buy than compatibility of team members. (Weinberg)

7 Zusätzliche Erkenntnisse

In den Interviews und bei der anschliessenden Ausarbeitung des vorhergehenden Kapitels hat sich auch gezeigt, dass die von mir vorgeschlagenen Grundsätze nur eine von verschiedenen möglichen Aufteilungen ist. Sie erweist sich aber als nützlich, um der gesamten Komplexität des Software Engineering eine Struktur zu geben.

Es ergaben sich einige weitere Punkte, die in verschiedenen Zusammenhängen als wichtig erwähnt wurden. In einem erweiterten Schema könnten diese auch als Grundsätze definiert werden.

7.1 Kommunikation

Aktive Kommunikation zwischen allen Beteiligten ist eine Grundvoraussetzung für erfolgreiche Projekte. Der beste Mitarbeiter nützt nichts, wenn, aus welchen Gründen auch immer, kein Informationsaustausch mit ihm stattfindet.

Die Wichtigkeit der Kommunikation gilt bei vielen Projekten und ist nicht auf das Gebiet Software Engineering beschränkt. Das trifft auch für die folgende Liste von Kritierien zum Thema Kommunikation zu:

- Sender und Empfänger: Informationsaustausch findet immer zwischen mindestens zwei Personen statt. Der Sender bringt eine Botschaft in eine übermittelbare Form, der Empfänger muss die Botschaft für sich wieder entschlüsseln. Diese Tatsache führt dazu, dass
  - bei der nötigen Umwandlung immer Information verloren geht. "Keine Form der Kommunikation ist perfekt und vollständig und in einem Softwareprojekt ist es unsere Aufgabe, diese Unvollständigkeit zu organisieren."[Cock99a]
  - der Sender nicht vollständig beeinflussen kann, wass der Empfänger versteht. Der Empfänger entscheidet mit, was kommuniziert wurde. Feedback in irgendeiner Form ist deshalb immer nötig.
- Form: Die effizienteste Form zu kommunizieren, sind zwei Personen, die einander gegenübersitzen. Am schwerfälligsten ist die Kommunikation in geschriebener Form (Papier). Dazwischen gibt es Abstufungen wie Telefon, Email, Newsgroups. Effizienz ist allerdings nicht das einzige Kriterium, dass die Form beeinflusst. Geschriebene Papiere bedeuten eine höhere Verbindlichkeit als mündliche Abmachungen, Email erlaubt eine zeit- und ortsunabhängige Kommunikation.

- "Man kann nicht nicht kommunizieren.": Kommunikation besteht keineswegs nur aus Worten, sondern aus Verhalten jeder Art wie Tonfall und Schnelligkeit der Sprache, sowie Körperhaltung und Ausdrucksbewegungen. Diese Art von Kommunikation ist schwer kontrollierbar und beeinflussbar, da sie stark von den jeweiligen Personen abhängt. Sie beeinflusst auch die bevorzugte Form der Kommunikation in einer Gruppe und damit schlussendlich wieder die Effizienz des Informationsaustausches.

7.2 Projektverantwortlicher

Ist demokratisches Software Engineering möglich? Brooks beantwortet diese Frage in seinem Essay "Aristocrazy, Democrazy and System Design"[Brook75a] mit "Ja und Nein".
"Ja" in dem Sinn, dass beim Entwurf eines grossen Systems nur wenig Leute mitreden sollen, damit dieses klare und einfache Züge erhält, analog beim Bau einer Kathedrale.
"Nein" in dem Sinn, dass Brooks erkannte, dass auch die Implementation eine kreative Arbeit ist und es sich bei den Entwicklern nicht um wenig denkende, nur ausführende Personen handelt.

In vielen Entwurfsmethoden wird versucht, Systeme so zu entwickeln, dass die Mitarbeiter mehr oder weniger gut austauschbare Ressourcen sind. Das geschieht mit dem Ziel, den Projektablauf und die Projektplanung besser kontrollieren zu können. Personen sind nun aber nicht beliebig austauschbar, so dass solche Methoden nur beschränkt funktionieren können.

Ich denke, jedes Projekt braucht einen Projektverantwortlicher, welcher eine Vorstellung des Endresultates hat. Demokratie beim Entwerfen ist nur beschränkt möglich, wie uns das Beispiel aus der Architektur mit der Kathedrale zeigt.

Der wichtigste Grund für diesen Schluss sehe ich darin, dass wie der Name schon sagt, in jedem Projekt jemand die Verantwortung für das Resultat übernehmen muss. Oder anders gesagt, eine Vision haben muss und alle Entscheide in einem Projekt prüft, ob sie diese Vision unterstützen oder ihr widersprechen.

Wenn alle verantwortlich sind ist schlussendlich niemand verantwortlich. Verantwortlich sein heisst auch, Kritik von Personen aus dem Projekt aufzunehmen und zu berücksichtigen. Aber Designentscheide werden schlussendlich von einer Person gefällt.

Bei grösseren Projekten gibt es das Prinzip des Verantwortlichen auf verschiedenen Stufen. Jemand ist verantwortlich für die Systemarchitektur und die Aufteilung in einzelne Applikationen ohne sich um jedes Detail zu kümmern. Andere wiederum sind für eine Teilapplikation verantwortlich.

Ein mögliches Problem mit "Verantwortlichen" von Projekten oder Teilen davon ist, dass diese Personen das Endresultat zu individuell prägen. Feedback und Kritik sind hier wichtig. Verantwortlich heisst nicht, "Es geht niemand sonst etwas an". Verantwortung heisst vor allem, "Ich sorge dafür, dass dieser Teil erfolgreich realisiert wird".

Die Idee von Projektverantwortlichen steht in einem gewissen Widerspruch zur Forderung des "Collective Ownership of Code" beim Extreme Programming. Eine kleine Gruppe kann

gemeinsam die Verantwortung für etwas übernehmen, aber dann muss es sich bei der Gruppe um ein sehr gut aufeinander abgestimmtes und eingespieltes Team handeln.

Wie bei vielen Regeln im Software Engineering gibt es auch hier nicht die eine Wahrheit, aber Anonymität in einem Projekt birgt stark die Gefahr, dass nicht sauber gearbeitet wird und fördert sicher nicht, dass sauberer Code entsteht.

7.3 Offene Fragen

Während den Interviews sind mir ein paar Fragen aufgefallen, welche ich mit den hier zusammengetragenen Informationen und auch aus eigener Erfahrung nur schwer beantworten kann:

- Warum werden heute noch immer sehr häufig zu enge Zeitpläne aufgestellt? Auch hier ist es vielleicht schwierig, eine wissenschaftlich fundierte Erklärung zu finden. Am ehesten sehe ich die Gründe in dem was De Marco[DeMar99a] als "Management nach der spanischen Theorie" bezeichnet. Für viel Manager bedeutet Produktivität nur, dass man versucht, mehr aus einem Tageslohn herauszuschinden.
- Warum werden sehr selten automatisierte Tests angewendet? Es wird überall getestet, mehr oder weniger intensiv, aber die meisten Testverfahren werden manuell geführt durch eine Testperson. Sind automatisierte Tests zu aufwendig in der Herstellung? Vertrauen wir zuwenig auf automatisierte Tests? Ich vermute den Grund vor allem hier. Ein automatisierter Test kann nur eine 0/100% Antwort geben und nie eine Antwort in der Form, dass das Programm stabil genug läuft.
- Warum werden selten Codereviews durchgeführt? Das hat vermutlich auch verschiedene Gründe. Sourcecode ist eine persönliche Arbeit und viele Leute können nicht mit Kritik umgehen, ohne diese Kritik auf sich als Person zu beziehen. Und noch viel schwieriger ist es, solche Arbeiten in einer konstruktiven Art und Weise zu kritisieren.

8 Schlussfolgerungen

Ziel meiner Arbeit war es, auf möglichst einfache und klare Art zu formulieren, was die wichtigsten Gründe für erfolgreiche Softwareprojekte sind.

Rückblickend auf zahlreiche gelesene Bücher und Artikel, auf die geführten Interviews mit verschiedenen Personen in verschiedenen Firmen komme ich zum ernüchternden Schluss, dass die möglichen Probleme komplexer und vielschichtiger sind, als ich zu Beginn meiner Arbeit auch nur erahnte. Das ist vor allem der Fall, sobald man eine relativ allgemeine Sicht verlässt und sich mit Detailproblemen eines einzelnen Projektes auseinandersetzt. Und es sind schlussendlich immer diese Details die über Erfolg oder Misserfolg entscheiden.

Die Resultate der Interviews lassen gewisse Schlüsse in einzelnen Projekten zu, welche aber häufig nur schwer verallgemeinert werden können. Insgesamt gesehen lassen sich auch Trends erkennen wie dass Entwickler Qualität hoch gewichten, Projektleiter und Manager diese zu Gunsten des Zeitplanes auch mal zurückstellen. Aber daraus ergeben sich noch keine verbindlichen Regeln.

Dass trotz vielen Problemen der grosse Teil der Projekte mehr oder weniger erfolgreich abgeschlossen wurden, lässt sich also nicht auf einen einfachen Nenner bringen. Die erfolgreiche Methode gibt es nicht. In allen Projekten waren schlussendlich die beteiligten Personen der ausschlaggebende Erfolgsfaktor, manchmal vielleicht sogar trotz der angewandten Methoden.

Für erfolgreiche Projekte braucht man also nicht primär die richtigen Methoden, sondern die richtigen Personen. Damit ist noch kein Problem gelöst aber es ist definiert, wo der primäre Ansatzpunkt ist, um bei der Softwareentwicklung weiterzukommen: bei den beteiligten Personen. Die Ausbildung hat damit eine zentrale Bedeutung, wobei es sich dabei um eine Ausbildung des Handwerks "Softwareentwicklung" und nicht eine Ausbildung des "Wissens Sammelns" handelt. Für solche Ausbildungen sind rein schulorientierte Umfelder wie Universitäten und Fachhochschulen nur bedingt geeignet.

Alistair Cockburn und Gerald Weinberg haben diese Einsicht sehr treffend begründet, weshalb ich die wichtigsten Absätze dazu hier zitiere.

Cockburn schreibt[Cock99a]: *It appears to me that software development is happening in industry, not in the universities. Universities are great for problems that can be solved by sitting alone and thinking or experimenting for months on end. Universities were great for giving us automata theory, complexity analysis, compilers and the like. But universities are not at all well suited to understanding what is happening during software development.*
Software development -at the moment- is much more like early manufacture of samurai swords, shields and battlefield tactics. You make a pile of swords or war tactics, send them onto the battlefield, and see which ones worked better. Then you make different swords and tactics, and so on. You can't figure out the right answer sitting alone in the room. You have to be on the battlefield. I can't imagine learning the things I've learnt while sitting peacefully in my office reflecting. Most of my original reflections and predictions were just wrong. So any one of you who is interested in this topic probably has to work as a developer or consultant, so you can see the moment-to-moment action and get raw data.

Weinberg schreibt[Wein71a]: *After 25 years of learning, I don't think programming is too complex a behavior to be studied, but it may be too complex to be studied in a laboratory. … Over the years, I've observed that the best programmers are the most introspective. If they do something wrong, they examine the mental processes that led to the problem; then, they do something to change the process. … Many college-trained developers approach a project as they would a one-semester class assignment. They think about it or ignore it for a while, then suddenly start jamming on the keyboard to produce something that will earn a passing grade. Software projects done at the universities generally don't have to be maintainable, usable or testable by another person.*

Das Wissen und Kennen von Methoden, Programmiersprachen und Theorien ist ein wichtiger, aber nur erster kleiner Schritt zum Erfolg. Erst bei der Umsetzung in realen Projekten, macht man die Erfahrungen, die nötig sind, um Softwareprojekte auch wirklich erfolgreich zu Ende zu führen. Es ist ein dauerndes Wechselspiel von Wissen aneignen und Umsetzung in der Praxis, wobei der Schwerpunkt klar beim zweiten liegen muss.

Wo die beteiligten Personen das wichtigste Element darstellten, spielen persönliche Meinungen und Wahrnehmungen eine wichtige Rolle. Diese lassen sich nur schwer eine Form analog zu naturwissenschaftlichen Gesetzen bringen. Viel geeigneter sind Werte und Prinzipien als Beschreibung einer Menge übereinstimmender Meinungen. Genau das hat vor kurzem eine Gruppe namhafter Softwareentwickler und Methodiker gemacht und in meiner Arbeit wurde ich praktisch durchwegs bestätigt, dass dies der richtige Ansatz zum Erfolg ist:

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over process and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Eine weitere Erkenntnis und auch Bestätigung dafür, dass der Weg zu erfolgreicher Software "by doing it" ist, war meine Erfahrung, dass beinahe alles bereits in irgend einer Form irgendwo formuliert wurde. Eine kurze Recherche im Internet hat gezeigt, dass mehr oder weniger jeder Tipp den ich jeweils unter "Umsetzung in der Praxis" formuliert habe, schon mehrfach anderswo niedergeschrieben wurde.

Das wichtigste Resultat dieser Arbeit ist somit die persönliche Erfahrung beim Formulieren einer eigenen Sicht auf die Probleme bei der Softwareentwicklung. Leser dieser Aufzeichnungen können davon sicher auch profitieren, aber viel wichtiger sind eigene persönliche Erfahrungen in Softwareprojekten "by doing it".

9 Literaturverzeichnis

9.1 Bücher und Publikationen

[Alex77a] Christopher Alexander, Sara Ishakawa and Murray Silverstein, *A Pattern Language*, Oxford University Press, 1977

[Beck99a] Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999

[Boe88a] B. Boehm, *A Spiral Model for Software Development and Enhancement*, Computer, vol. 21, no. 5, May 1988

[Brook75a] Frederick P. Brooks, *The Mythical Man-Month*, Addison-Wesley, 1975.

[Brown98a] William H. Brown, Raphael C. Malveau, Hays W. McCormick, Thomas J. Mowbray, *Anti Patterns*, Wiley Computer Publishing, 1998

[Cantu99a] Marco Cantù, *When RAD is bad*, 1999, http://community.borland.com/devnews/article/0,1714,10463,00.html

[Chaos94a] The Standish Group, *The CHAOS Report*, 1994 http://www.pm2go.com/sample_research/chaos_1994_1.asp

[Cock99a] Alistair Cockburn, *Software Development as a Cooperative Game*, http://www.members.aol.com/humansandt/papers/asgame/asgame.htm

[Cock99b] Alistair Cockburn, *Characterizing People as Non-Linear, First-Order Components of Software Development*, http://www.members.aol.com/humansandt/papers/nonlinear/nonlinear.htm

[DeMar99a] Tom DeMarco and Timothy Lister, *Peopleware - Productive Projects and Teams. 2nd ed.*, Dorset house Publishing Co., 1987/99

[Dere76a] Frank DeRemer and Hans H. Kron, *Programming In the Large Versus Programming In the Small*, IEEE Transactions on Software Engineering, 1976

[Duca02a] Stéphane Ducasse, Serge Demeyer, Oscar Nierstrasz, *Object-Oriented Reengineering Patterns*, 2002

[Fow97a] Martin Fowler, UML Distilled, *Applying the Standard Object Modeling Language*, Addison-Wesley, 1997

[Fow01a] Martin Fowler, *The New Methodology*, http://www.martinfowler.com/articles/newMethodology.htm

[Gabor91a] Pintér Gábor, *To comment or not to comment*, http://www.community.borland.com/article/0,1410,26678,00.html, 2001

[Gamm95a] Erich Gamma, Richard Helm, John Vlissides and Ralph E. Johnson, *Design Patterns*, Addison-Wesley, 1995

[Gold95a] Adele Goldberg and Kenneth S. Rubin, *Succeeding with Objects*, Addison-Wesley, 1995

[Hermes] BFI - Bundesamt für Informatik, *HERMES: Führung und Abwicklung von Informatikprojekten*, 1995
Hermes Online: http://www.isb.admin.ch/dok/hermes.htm

[Hum95a] Watts S. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley, 1995

[Hunt99a] Andrew Hunt and David Thomas, *The Pragmatic Programmer*, Addison-Wesley, 1999/2000

[Jac92a] I. Jacobson, *Object-Oriented Software Engineering*, Addison-Wesley, 1992

[McI69a] Doug McIlroy, *Mass Produced Software Components*, in Proceedings of NATO conference on Software Engineering, Petrocelli/Charter, New York, 1969

[Mey88a] Bertrand Meyer, *Object-Oriented Software Construction*, Prentice Hall, 1988

[Mey93a] Bertrand Meyer, *An introduction to Design by Contract* http://www.eiffel.com/doc/manuals/technology/contract/

[Noa99a] Jörg Noack, Bruno Schienmann, *Objektorientierte Vorgehensmodelle im Vergleich*, Informatik-Spektrum, Springer-Verlag, 22/1999

[Pres97a] Roger S. Pressman, *Software Engineering, A practitioner's approach*, The McGraw-Hill Companies, 1997

[Rum95a] James Rumbaugh, *Modeling & Design, What is a method ?*, 1995 http://www.rational.com/products/whitepapers/396.jsp#p5

[SW98a] D. D'Souza, Alan Wills, *Objects, Components, and Frameworks with UML – the Catalysis Approach*. Addison-Wesley, 1993

[Stan94a] Standish Group, *The CHAOS Report*, 1994 http://www.pm2go.com/sample_research/chaos_1994_1.asp

[Tra95a] Will Tracz, *Confession of a Used Program Salesman*, Addison-Wesley, 1995

[Vleck89a] Tom Van Vleck, *Three questions about each bug you find*, 1998 http://www.multicians.org/thvv/threeq.html

[Wein71a] Gerald M. Weinberg, *The Psychology of Computer-Programming (Silver Anniversary Edition)*, Dorset House, 1971 / 1998

[WiWi90a] Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener, *Designing Object-Oriented Software*, Prentice Hall, 1990

9.2 Verschiedene Links

Homepage Alistair Cockburn, Salt Lake City, USA
http://members.aol.com/acockburn

Homepage Martin Fowler, Chicago, USA
http://www.martinfowler.com

Homepage Gerald M. Weinberg, USA
http://www.geraldmweinberg.com

Wirfs Brock Associates, Responsibility-Driven Design
http://www.wirfs-brock.com

Manifesto for Agile Software Development
http://www.agilealliance.org

International Organization for Standardization
http://www.iso.ch

Capability Maturity Model for Software
http://www.sei.cmu.edu/cmm

Extreme Programming Resource
http://www.xprogramming.com

Extreme Programming: A gentle introduction
http://www.extremeprogramming.org

"Standish Group", West Yarmouth, USA
http://www.pm2go.com/

Homepage SUMit, based in Gouda, Holland
http://www.sum-it.nl

Software Build and Fix,by J. Adrian Zimmer
http://www.mapfree.com/sbf/

A computer user's manifesto
http://www.businessweek.com/1998/39/b35970337.htm

Anhang A - Fragebogen

- Fragebogen Version erste Befragung
- Fragebogen Version zweite Befragung
- Hinweise, welche die befragten Personen vorgängig erhielten

Fragebogen

Im Rahmen meiner Diplomarbeit an der Universität Bern habe ich eine Reihe von Thesen aufgestellt, die zur erfolgreichen Abwicklung eines Softwareprojektes führen sollen. Mit Hilfe von Fallbeispielen sollen diese Thesen nun bestätigt werden.

Damit eine sinnvolle Auswertung gemacht werden kann, sollten Sie bei der Fragebeantwortung folgende Punkte beachten:

- Der Fragebogen ist so aufgebaut, dass zu jeder These eine Reihe von Fragen vorhanden sind.
- Beantworten Sie die Fragen bitte nicht allgemein sondern immer bezogen auf das ausgewählte Projekt.
- Die Antworten sollen Ihrer persönlichen Meinung und nicht der generell gültigen Meinung des Projektteams entsprechen.

Allgemeine Angaben zum Projekt

Name: .....

Firma (freiwillig): .....

Projektbezeichnung (freiwillig): .....

Ihre Rolle im Projekt: ❑ Anwender ❑ Projektleiter ❑ Entwickler

Branche: .....

Fachgebiete:
❑ Business Software
❑ Wissenschaftliche Software
❑ Internet Applikation
❑ Hardware Programmierung
❑ Systemsoftware

Zeitdauer des Projektes: Beginn am:..... Ende am:.....

Anzahl beteiligte Personen: Total: ..... davon Entwickler: .....

Aufwand in Arbeitstagen: Total: ..... davon Entwicklung: .....

Umfang des Projektes: Anzahl Applikationen: .....

Anzahl Zeilen Sourcecode: dieses Projekt:..... eigene Library:..... Drittlibraries: .....

Anzahl Klassen: dieses Projekt:..... eigene Library:..... Drittlibraries: .....

Programmiersprache(n): .....

Entwicklungsumgebung(en): .....

These 1: Einbezug Benutzer

> Der Benutzer muss zu jedem Zeitpunkt vollständig in die Projektabwicklung integriert sein. Spezifikation und Diskussion muss in der Sprache des Benutzers stattfinden. Dazu geeignete Hilfsmittel sind Skizzen von Bildschirmmasken und Reports, einfache Ablaufdiagramme, Checklisten und Prototypen. Es empfiehlt sich, für jedes Projekt, einen "Key User" ins Entwicklungsteam aufzunehmen, welcher mit der Problemstellung und den Abläufen sehr gut vertraut ist.

Im betrachteten Projekt wurde die These 1 erfüllt:

❑ gar nicht ❑ kaum ❑ mittelmässig ❑ ziemlich ❑ ausserordentlich

Wer hat die Spezifikation geschrieben?

❑ Anwender ❑ Projektleiter ❑ Entwickler ❑ Drittfirma ❑ .....

In welcher Form wurde die Spezifikation erstellt:

.....

.....

Wurde die Spezifikation im Verlaufe des Projektes angepasst?

❑ nie ❑ selten ❑ gelegentlich ❑ oft ❑ immer

Wann erhielten die Anwender erste Programmversionen? Anteil Projektgesamtdauer .....

Wie häufig erhielten die Anwender neue Versionen?

❑ einmal bei Projektende ❑ Vorabversion und Endversion ❑ monatlich ❑ wöchentlich ❑ täglich

Gab es eine Kontrolle auf Einhaltung der Spezifikation bei Inbetriebnahme von neuen Versionen?

❑ nie ❑ selten ❑ gelegentlich ❑ oft ❑ immer

In welcher Form wurde die Kontrolle durchgeführt:

.....

.....

.....

Ich finde die These 1:

❑ völlig falsch ❑ ziemlich falsch ❑ unentschieden ❑ ziemlich richtig ❑ völlig richtig

Weitere Bemerkungen:

.....

.....

.....

.....

.....

These 2: Einfach und klein

> Sowohl die Benutzerschnittstelle als auch der Programmcode sollen einfach sein. Nur konsequentes Refactoring und ständige Kontrolle auf Code Duplikation können das garantieren. Entwicklerteams sollen nie mehr als zwei bis drei Personen umfassen, damit die nötige Kommunikation auf ein Minimum beschränkt wird. Grössere Projekte sollen in Teilprojekte aufgeteilt werden.

Im betrachteten Projekt wurde die These 2 erfüllt:

❑ gar nicht ❑ kaum ❑ mittelmässig ❑ ziemlich ❑ ausserordentlich

Wird das Endprodukt aktiv eingesetzt?

❑ nie ❑ selten ❑ gelegentlich ❑ oft ❑ immer

Gibt es Funktionalitäten in der Software, die selten oder nie gebraucht werden?

❑ keine ❑ wenig ❑ normal ❑ viel ❑ sehr viel

Falls unötige Funktionalitäten vorhanden sind, warum?

.....

.....

Was ist der Nutzen der Software?

❑ Zeiteinsparung ❑ höhere Qualität/ Genauigkeit ❑ zusätzlich verfügbare Infos ❑ strukturiertere Arbeitsabläufe ❑ .....

Wieviel Schulung ist nötig, um mit dem Produkt zu arbeiten?

❑ keine ❑ wenig ❑ normal ❑ viel ❑ sehr viel

Wieviele Projektmitglieder kennen den gesamten Sourcecode?

❑ niemand ❑ einer ❑ mehrere ❑ fast alle ❑ alle

War es aufwendig, nach Abschluss aufgetretene Fehler zu korrigieren?

❑ gar nicht ❑ kaum ❑ mittelmässig ❑ ziemlich ❑ ausserordentlich

Ich finde die These 2:

❑ völlig falsch ❑ ziemlich falsch ❑ unentschieden ❑ ziemlich richtig ❑ völlig richtig

Weitere Bemerkungen:

.....

.....

.....

.....

.....

These 3: Qualität in allen Teilen

> Jeder Teil in einem Projekt soll in der jeweils höchst möglichen Qualität realisiert werden. Dazu gehört eine sorgfältige Planung am Anfang, sauberer Programmcode (auch für Testprogramme und Prototypen), sowie die Vollständigkeit von allen Teilen beim Projektende. Grundvoraussetzung für hohe Qualität ist ein breit abgestütztes Basiswissen von allen Projektmitarbeitern.

Im betrachteten Projekt wurde die These 3 erfüllt:

❑ gar nicht ❑ kaum ❑ mittelmässig ❑ ziemlich ❑ ausserordentlich

Gibt es im Endprodukt bekannte, nicht behobene Fehler?

❑ keine ❑ wenig ❑ normal ❑ viele ❑ sehr viele

Gibt es Teile in der Software die neu geschrieben werden sollten?

❑ keine ❑ wenig ❑ normal ❑ viele ❑ sehr viele

Wenn ja warum ?

.....

.....

Wieviele Fehler sind nach Abgabe der Software aufgetaucht ?

❑ keine ❑ wenig ❑ normal ❑ viele ❑ sehr viele

Wurden regelmässig Tests durchgeführt?

❑ keine ❑ wenig ❑ normal ❑ viele ❑ sehr viele

In welcher Form wurde getestet ?

.....

.....

Welche Art von Dokumentationen wurden erstellt ?

.....

.....

Ist die Software vollständig dokumentiert? ❑ ja ❑ nein ❑ teilweise

Ist die Benutzerdokumentation vollständig? ❑ ja ❑ nein ❑ teilweise

Ich finde die These 3:

❑ völlig falsch ❑ ziemlich falsch ❑ unentschieden ❑ ziemlich richtig ❑ völlig richtig

Weitere Bemerkungen:

.....

.....

.....

.....

.....

These 4: Zeitplan

Der zeitliche Ablauf eines Projektes soll klar geplant sein. Dies umfasst eine Gesamtplanung von Start bis Ende, als auch eine Planung für die einzelne Woche und Arbeitstag. Für jede Zeiteinheit muss ein klares Ziel festgelegt werden. Die Einhaltung der Ziele muss laufend kontrolliert werden. Die Kontrolle führt immer zu Anpassungen am Zeitplan und/oder den Zielen.

Im betrachteten Projekt wurde die These 4 erfüllt:

❑ gar nicht ❑ kaum ❑ mittelmässig ❑ ziemlich ❑ ausserordentlich

Wurden Milestones für das Projekt definiert?

❑ nie ❑ selten ❑ gelegentlich ❑ oft ❑ immer

Milestones Anzahl ..... zeitlicher Abstand.....

Wie wurden die Ziele festgelegt? ❑ gar nicht ❑ schriftlich ❑ mündlich

Präzisierung zur Form der Ziele:

.....

.....

.....

Waren jedem Mitarbeiter der Zeitplan/Ziele bekannt?

❑ keinesfalls ❑ wahrscheinlich nicht ❑ vielleicht ❑ ziemlich wahrscheinlich ❑ ganz sicher

Wurden die Milestones eingehalten ?

❑ nie ❑ selten ❑ gelegentlich ❑ oft ❑ immer

Wie wurde die Einhaltung von Milestones überprüft?

.....

.....

.....

Ich finde die These 4:

❑ völlig falsch ❑ ziemlich falsch ❑ unentschieden ❑ ziemlich richtig ❑ völlig richtig

Weitere Bemerkungen:

.....

.....

.....

.....

.....

These 5: Spass der Beteiligten

> Software Entwicklung ist eine kreative Tätigkeit. Bei solchen Arbeiten ist die Produktivität stark davon abhängig, ob die Beteiligten mit Freude und Begeisterung mitmachen. Voraussetzungen dafür sind ein angenehmes Arbeitsklima (räumlich / personell) und eine persönliche Identifikation mit der Aufgabenstellung der Projektmitarbeiter.

Im betrachteten Projekt wurde die These 5 erfüllt:

❑ gar nicht ❑ kaum ❑ mittelmässig ❑ ziemlich ❑ ausserordentlich

Was war die wichtigste Motivation unter den Projektmitarbeitern?

❑ Geld/Lohn ❑ Mitarbeit beim Projekt ❑ Inhalt/Thema des Projektes ❑ Position im Projektteam ❑ .....

Gab es im Laufe des Projektes Konflikte unter den Beteiligten?

❑ nie ❑ selten ❑ gelegentlich ❑ oft ❑ immer

Wurden aufgetretene Konflikte gelöst?

❑ nie ❑ selten ❑ gelegentlich ❑ oft ❑ immer

Treffen sich die Projektbeteiligten auch zu Anlässen neben der Arbeit?

❑ nie ❑ selten ❑ gelegentlich ❑ oft ❑ immer

Gab es viele personelle Wechsel im Projektteam?

❑ gar nicht ❑ kaum ❑ mittelmässig ❑ ziemlich ❑ ausserordentlich

Was waren die Gründe für personelle Wechsel?

.....

.....

Ich finde die These 5:

❑ völlig falsch ❑ ziemlich falsch ❑ unentschieden ❑ ziemlich richtig ❑ völlig richtig

Weitere Bemerkungen:

.....

.....

.....

.....

.....

War das Projekt erfolgreich?

Die Termine wurden eingehalten:

❑ gar nicht ❑ kaum ❑ mittelmässig ❑ ziemlich ❑ ausserordentlich

Der Kostenrahmen wurde eingehalten:

❑ gar nicht ❑ kaum ❑ mittelmässig ❑ ziemlich ❑ ausserordentlich

Die Software ist fehlerfrei:

❑ keinesfalls ❑ wahrscheinlich nicht ❑ vielleicht ❑ ziemlich wahrscheinlich ❑ ganz sicher

Die Software erfüllt die Benutzeranforderungen:

❑ keinesfalls ❑ wahrscheinlich nicht ❑ vielleicht ❑ ziemlich wahrscheinlich ❑ ganz sicher

Die Software ist wartbar (hinreichend dokumentiert, einfach anzupassen):

❑ keinesfalls ❑ wahrscheinlich nicht ❑ vielleicht ❑ ziemlich wahrscheinlich ❑ ganz sicher

Folgende Rahmenbedingungen machten das Projekt speziell einfach:

.....

.....

.....

.....

.....

Folgende Rahmenbedingungen machten das Projekt speziell kompliziert:

.....

.....

.....

.....

.....

Weitere Bemerkungen:

.....

.....

.....

.....

.....

Fragebogen / Interview

Im Rahmen meiner Diplomarbeit an der Universität Bern habe ich eine Reihe von Grundsätzen aufgestellt, die zur erfolgreichen Abwicklung eines Softwareprojektes führen sollen. Mit Hilfe von Fallbeispielen sollen diese Grundsätze nun überprüft werden. Es handelt sich um folgende fünf Grundsätze:

1) Einbezug Benutzer: Der Benutzer muss zu jedem Zeitpunkt vollständig in die Projektabwicklung integriert sein. Spezifikation und Diskussion muss in der Sprache des Benutzers stattfinden. Dazu geeignete Hilfsmittel sind Skizzen von Bildschirmmasken und Reports, einfache Ablaufdiagramme, Checklisten und Prototypen. Es empfiehlt sich, für jedes Projekt, einen "Key User" ins Entwicklungsteam aufzunehmen, welcher mit der Problemstellung und den Abläufen sehr gut vertraut ist und der auch in der Lage ist, Wünsche und Anpassungen verständlich zu formulieren.

2) Einfach und klein: Sowohl die Benutzerschnittstelle als auch der Programmcode sollen einfach sein. Nur konsequentes Refactoring und ständige Kontrolle auf Code Duplikation können das garantieren. Entwicklerteams sollen nie mehr als zwei bis drei Personen umfassen, damit die nötige Kommunikation auf ein Minimum beschränkt wird. Grössere Projekte sollen in Teilprojekte aufgeteilt werden. In diesem Fall sollen auch die Schnittstellen zwischen Teilprojekten möglichst einfach und klein sein.

3) Qualität in allen Teilen: Jeder Teil in einem Projekt soll in der jeweils höchst möglichen Qualität realisiert werden. Dazu gehört eine sorgfältige Planung am Anfang, sauberer Programmcode (auch für Testprogramme und Prototypen), sowie die Vollständigkeit von allen Teilen beim Projektende. Grundvoraussetzung für hohe Qualität ist ein breit abgestütztes Basiswissen von allen Projektmitarbeitern.

4) Zeitplan: Der zeitliche Ablauf eines Projektes soll klar geplant sein. Dies umfasst eine Gesamtplanung von Start bis Ende, als auch eine Planung für die einzelne Woche und den einzelnen Arbeitstag. Für jede Zeiteinheit muss ein klares Ziel festgelegt werden. Die Einhaltung der Ziele muss laufend kontrolliert werden. Die Kontrolle führt immer zu Anpassungen am Zeitplan und/oder den Zielen.

5) Spass der Beteiligten: Software Entwicklung ist eine kreative Tätigkeit. Bei solchen Arbeiten ist die Produktivität stark davon abhängig, ob die beteiligten Personen mit Freude und Begeisterung mitmachen. Voraussetzungen dafür sind ein angenehmes Arbeitsklima (räumlich / personell) und eine persönliche Identifikation der Projektmitarbeiter mit der Aufgabenstellung.

Damit eine sinnvolle Auswertung gemacht werden kann, sollten Sie bei der Fragebeantwortung folgende Punkte beachten:

- Beantworten Sie die Fragen nicht allgemein sondern immer bezogen auf das ausgewählte Projekt.
- Die Antworten sollen Ihrer persönlichen Meinung und nicht der generell gültigen Meinung des Projektteams entsprechen. Sämtliche Aussagen werden nur in anonymisierter Form weiterverwendet. Die Tonbandaufnahmen der Interviews werden nach der Auswertung gelöscht.
- Bei Fragen, mit Skala von 1-7 haben die Werte folgende Bedeutung:
  1 gar nicht oder nie
  4 durchschnittlich oder mittelmässig
  7 ausserordentlich oder immer

Aufbau der Fragen

- Im ersten Teil des Interviews werden Ihnen allgemeine Fragen zum Projekt und zur gewählten vorgehensweise gestellt.
- Im zweiten Teil beziehen sich die Fragen jeweils auf einen der Grundsätze und wieweit dieser im Projekt berücksichtigt wurde.

Name: .....

Firma: .....

Allgemeine Angaben zum Projekt

Projektbezeichnung: .....

Ihre Rolle im Projekt: ❑ Anwender ❑ Projektleiter ❑ Entwickler

Fachgebiet: ❑ Business Software ❑ Hardware Programmierung
❑ Wissenschaftliche Software ❑ Systemsoftware
❑ Internet Applikation

Zeitdauer des Projektes: Beginn am:..... Ende am:.....

Anzahl beteiligte Personen: Total: ..... davon Entwickler: .....

Aufwand in Arbeitstagen: Total: ..... davon Entwicklung:.....

Umfang des Projektes: Anzahl Applikationen: .....

Anzahl Zeilen Sourcecode: dieses Projekt:..... eigene Library:..... Drittlibraries:.....

Anzahl Klassen: dieses Projekt:..... eigene Library:..... Drittlibraries:.....

Programmiersprache(n): .....

Entwicklungsumgebung(en): .....

War das Projekt erfolgreich?

Termine wurden eingehalten:	❑ 1	❑ 2	❑ 3	❑ 4	❑ 5	❑ 6	❑ 7
Kostenrahmen wurde eingehalten:	❑ 1	❑ 2	❑ 3	❑ 4	❑ 5	❑ 6	❑ 7
Software ist fehlerfrei:	❑ 1	❑ 2	❑ 3	❑ 4	❑ 5	❑ 6	❑ 7
Software erfüllt die Benutzeranforderungen:	❑ 1	❑ 2	❑ 3	❑ 4	❑ 5	❑ 6	❑ 7
Software ist wartbar: *(hinreichend dokumentiert, einfach anzupassen)*	❑ 1	❑ 2	❑ 3	❑ 4	❑ 5	❑ 6	❑ 7

? Welche Rahmenbedingungen machten das Projekt speziell einfach?

? Welche Rahmenbedingungen machten das Projekt speziell kompliziert?

? Was ist der Nutzen der erstellten Software?

? Wurde der Nutzen quantifiziert? Wenn ja, in welcher Form? Wenn nein, warum nicht?

Wie wurde das Projekt abgewickelt?

? Nach welcher Methode wurde vorgegangen (z.B. Extreme Programming)?

? Wie wurden die Projektmitarbeiter auf die verwendete Methode vorbereitet?

? Wie wurde geprüft, ob die Methode richtig angewendet wurde?

? Wurden spezielle Werkzeuge verwendet?

Grundsatz 1: Einbezug Benutzer

Der Benutzer muss zu jedem Zeitpunkt vollständig in die Projektabwicklung integriert sein. Spezifikation und Diskussion muss in der Sprache des Benutzers stattfinden. Dazu geeignete Hilfsmittel sind Skizzen von Bildschirmmasken und Reports, einfache Ablaufdiagramme, Checklisten und Prototypen. Es empfiehlt sich, für jedes Projekt, einen "Key User" ins Entwicklungsteam aufzunehmen, welcher mit der Problemstellung und den Abläufen sehr gut vertraut ist und der auch in der Lage ist, Wünsche und Anpassungen verständlich zu formulieren.

? Wurde dieser Grundsatz eingehalten? ❑ 1 ❑ 2 ❑ 3 ❑ 4 ❑ 5 ❑ 6 ❑ 7

? Wenn nein: Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

? Wenn ja: Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

? Wer hat die Spezifikation geschrieben?

❑ Anwender ❑ Projektleiter ❑ Entwickler ❑ Drittfirma ❑ .....

? In welcher Form wurde die Spezifikation erstellt?

? Wurde die Spezifikation im Verlaufe des Projektes angepasst und wenn ja, warum?

? Wann erhielten die Anwender erste Programmversionen? Anteil Projektgesamtdauer .....

? Wie häufig erhielten die Anwender neue Versionen?

? Wie wurde überprüft, ob die Spezifikation eingehalten wurde?

Ich finde den Grundsatz "Einbezug Benutzer"
wichtig für das Gelingen eines Projektes: ❑ 1 ❑ 2 ❑ 3 ❑ 4 ❑ 5 ❑ 6 ❑ 7

? Können Sie Ihre Gewichtung begründen ?

? Haben Sie weitere Bemerkungen zum Grundsatz "Einbezug Benutzer"?

Grundsatz 2: Einfach und klein

> Sowohl die Benutzerschnittstelle als auch der Programmcode sollen einfach sein. Nur konsequentes Refactoring und ständige Kontrolle auf Code Duplikation können das garantieren. Entwicklerteams sollen nie mehr als zwei bis drei Personen umfassen, damit die nötige Kommunikation auf ein Minimum beschränkt wird. Grössere Projekte sollen in Teilprojekte aufgeteilt werden. In diesem Fall sollen auch die Schnittstellen zwischen Teilprojekten möglichst einfach und klein sein.

? Wurde dieser Grundsatz eingehalten? ❑ 1 ❑ 2 ❑ 3 ❑ 4 ❑ 5 ❑ 6 ❑ 7

? Wenn nein: Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

? Wenn ja: Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

? Enthält das Programm Funktionen, die nie gebraucht werden?

? Falls unötige Funktionalitäten vorhanden sind, warum?

? Wieviel Schulung ist nötig, um mit dem Produkt zu arbeiten?

? Wie wurde die Arbeit unter den Projektmitarbeitern aufgeteilt?

? Was waren die Gründe für die gewählte Aufteilung?

? Führte die gewählte Aufteilung zu Problemen? Wenn ja, welche?

? Wieviele Projektmitglieder kennen den gesamten Sourcecode?

Ich finde den Grundsatz "Einfach und klein"
wichtig für das Gelingen eines Projektes: ❑ 1 ❑ 2 ❑ 3 ❑ 4 ❑ 5 ❑ 6 ❑ 7

? Können Sie Ihre Gewichtung begründen ?

? Haben Sie weitere Bemerkungen zum Grundsatz "Einfach und klein"?

Grundsatz 3: Qualität in allen Teilen

Jeder Teil in einem Projekt soll in der jeweils höchst möglichen Qualität realisiert werden. Dazu gehört eine sorgfältige Planung am Anfang, sauberer Programmcode (auch für Testprogramme und Prototypen), sowie die Vollständigkeit von allen Teilen beim Projektende. Grundvoraussetzung für hohe Qualität ist ein breit abgestütztes Basiswissen von allen Projektmitarbeitern.

? Wurde dieser Grundsatz eingehalten? ❑ 1 ❑ 2 ❑ 3 ❑ 4 ❑ 5 ❑ 6 ❑ 7

? Wenn nein: Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

? Wenn ja: Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

? Gibt es im Endprodukt bekannte, nicht behobene Fehler?

? Wenn ja: Warum sind die Fehler vorhanden?

? Wenn ja: Warum werden die Fehler nicht behoben?

? Wurden regelmässig Tests durchgeführt?

? In welcher Form wurde getestet?

? Gibt es Teile in der Software die neu geschrieben werden sollten?

? Wenn ja: Warum sollten welche Teile neu geschrieben werden?

? Gab es eine institutionalisierte Form für Design- und Codereview?

? Wenn ja: In welcher Form wurden die Reviews durchgeführt?

? Wenn nein: Warum fanden keine Reviews statt?

? Ist die Software hinreichend dokumentiert?

? Welche Art von Dokumentationen wurden erstellt?

Ich finde den Grundsatz "Qualität in allen Teilen"
wichtig für das Gelingen eines Projektes: ❑ 1 ❑ 2 ❑ 3 ❑ 4 ❑ 5 ❑ 6 ❑ 7

? Können Sie Ihre Gewichtung begründen ?

? Haben Sie weitere Bemerkungen zum Grundsatz "Qualität in allen Teilen"?

Grundsatz 4: Zeitplan

Der zeitliche Ablauf eines Projektes soll klar geplant sein. Dies umfasst eine Gesamtplanung von Start bis Ende, als auch eine Planung für die einzelne Woche und den einzelnen Arbeitstag. Für jede Zeiteinheit muss ein klares Ziel festgelegt werden. Die Einhaltung der Ziele muss laufend kontrolliert werden. Die Kontrolle führt immer zu Anpassungen am Zeitplan und/oder den Zielen.

? Wurde dieser Grundsatz eingehalten? ❑ 1 ❑ 2 ❑ 3 ❑ 4 ❑ 5 ❑ 6 ❑ 7

? Wenn nein: Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

? Wenn ja: Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

? Wie wurde der Zeitplan festgelegt? ❑ gar nicht ❑ schriftlich ❑ mündlich

? Wie detailliert war der Zeitplan aufgeteilt?

? Waren jedem Mitarbeiter der Zeitplan / Ziele bekannt?

? Wie wurde die Einhaltung Ziele / Zeitplan überprüft?

? Wie wurden auf Abweichungen Realität <-> Zeitplan reagiert?

Ich finde den Grundsatz "Zeitplan"
wichtig für das Gelingen eines Projektes: ❑ 1 ❑ 2 ❑ 3 ❑ 4 ❑ 5 ❑ 6 ❑ 7

? Können Sie Ihre Gewichtung begründen ?

? Haben Sie weitere Bemerkungen zum Grundsatz "Zeitplan"?

Grundsatz 5: Spass der Beteiligten

Software Entwicklung ist eine kreative Tätigkeit. Bei solchen Arbeiten ist die Produktivität stark davon abhängig, ob die beteiligten Personen mit Freude und Begeisterung mitmachen. Voraussetzungen dafür sind ein angenehmes Arbeitsklima (räumlich / personell) und eine persönliche Identifikation der Projektmitarbeiter mit der Aufgabenstellung.

? Wurde dieser Grundsatz eingehalten? ❑ 1 ❑ 2 ❑ 3 ❑ 4 ❑ 5 ❑ 6 ❑ 7

? Wenn nein: Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

? Wenn ja: Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

? Was war Ihre wichtigste Motivation bei der Arbeit an diesem Projekt? Warum?

? Gab es für sie persönlich wichtige Ziele, die nicht erreicht wurden? Welche? Warum?

? Was hat Ihnen bei diesem Projekt am wenigsten gefallen? Warum?

? Was hat bei diesem Projekt besonders gut funktioniert? Warum?

? Wenn Sie rückblickend Ihre Rolle im Projekt ändern könnten, was würden Sie ändern?

? Gab es im Laufe des Projektes Konflikte unter den beteiligten Personen?

? Wie wurden aufgetretene Konflikte gelöst?

? Gab es personelle Wechsel im Projektteam? Gründe ?

? Gab es gesellschaftliche Anlässe innerhalb des Projektteams? Welche? Von wem initiert?

? Nahmen Sie an solchen Anlässen teil? Warum? Warum nicht?

Ich finde den Grundsatz "Spass der Beteiligten"
wichtig für das Gelingen eines Projektes: ❑ 1 ❑ 2 ❑ 3 ❑ 4 ❑ 5 ❑ 6 ❑ 7

? Können Sie Ihre Gewichtung begründen ?

? Haben Sie weitere Bemerkungen zum Grundsatz " Spass der Beteiligten "?

Umfeld der Befragungen

Im Rahmen meiner Diplomarbeit als Informatiker an der Universität Bern, beschäftige ich mich mit der Frage, warum immer noch sehr viele Informatikprojekte fehlschlagen. Diese Tatsache ist eigentlich umso erstaunlicher, da sich in den letzten dreissig Jahren unzählige neue Tools, Prozesse und Methoden vorgeschlagen wurden.

Weiter deckt sich diese Tatsache nicht mit meinen persönlichen Erfahrungen in Informatikprojekten und ich versuche deshalb herauszufinden, was die Gründe sind, dass es in Projekten, die ich kenne, meistens gut bis sehr gut gelaufen ist.

Die Kombination von meinen persönlichen Erfahrungen, sowie dem theoretischen Wissen, dass ich mir im Studium angeeignet habe, versuchte ich in fünf Grundsätzen zusammenzufassen (-> siehe Fragebogen).

Warum Befragungen ?

Selbstbeobachtung allein genügt nicht, um etwas als Grundsatz zu formulieren. Die Befragungen sollen Hinweis darauf geben, wo die Grundsätze stimmen, wo nicht und wo allenfalls deren Grenzen liegen.

Ich denke, um die Probleme bei der Softwareentwicklung und dazu passende Lösungen herauszufinden, ist es richtig, Erfahrungen an Beispielen aus der Praxis anzuschauen.

Es reicht nicht theoretische Modelle aufzubauen. Softwareentwicklung ist eine Tätigkeit von Menschen für Menschen und so ist es durchausangebracht, eine klassische Methode aus der Verhaltungsforschung, wo Befragungen ein wichtiges Hilfsmittel sind, anzuwenden.

Ziel der Befragungen ?

Mit den Resultaten der Befragungen möchte ich zu jedem meiner Grundsätze detailliertere Informationen aufführen:

- Was sind die Bezüge zur Theorie und bestehenden Publikationen.
- Was sind mögliche Umsetzungen anhand von praktischen Beispielen (Teillösungen für Probleme).
- Was sind die Probleme eines Grundsatzes. Wo entstehen möglicherweise Widersprüche.

Anhang B - Antworten

Die hier aufgeführten Antworten stammen alle aus der zweiten Befragung.

Die Antworten der interviewten Personen sind jeweils pro Projekt zusammengefasst. Personen und Projekte wurden anonymisiert. Die Namen und Firmen werden nur mit dem Einverständnis der entsprechenden Personen bekannt gegeben.

Fragebogen / Interview Projekt 1

Allgemeine Angaben zum Projekt

Ihre Rolle im Projekt: *Entwickler*

Fachgebiet: *Systemsoftware*

Zeitdauer des Projektes: Beginn am:.....*Okt 2000* Ende am: .....*13. Sept 2001*
(Wichtige Anmerkung: das Projekt wurde teilweise "auf Eis" gelegt, um andere, dringendere Aufgaben zu erledigen – Feb + März 2001)

Anzahl beteiligte Personen: Total: .....*8* davon Entwickler: *3*

Aufwand in Arbeitstagen: Total: ..... *650* davon Entwicklung:.....*490*

Umfang des Projektes: Anzahl Applikationen:.....*5*

Anzahl Zeilen Sourcecode: dieses Projekt: . *48K* eigene Library: .*123K* Drittlibraries:..... *35K*

Anzahl Klassen: dieses Projekt: ...*112* eigene Library: ... *354* Drittlibraries:..... *70*

Programmiersprache(n): *Object Pascal, Transact-SQL*

Entwicklungsumgebung(en): *Borland Delphi 5 / MS SQL Server 7 / 2000*

War das Projekt erfolgreich?

Termine wurden eingehalten: *5*

Kostenrahmen wurde eingehalten: *6*

Software ist fehlerfrei: *6*

Software erfüllt die Benutzeranforderungen: *6*

Software ist wartbar: *4*

? Welche Rahmenbedingungen machten das Projekt speziell einfach?
Sehr gute Marktkenntnis vorhanden, Vorarbeiten (vorheriges Projekt für einen namhaften Kunden)

? Welche Rahmenbedingungen machten das Projekt speziell kompliziert?
Kein direkter Kunden-Kontakt, nicht "Software im Auftrag", sondern "shrink wrap software"

? Was ist der Nutzen der erstellten Software?
Ermöglicht Migration von Benutzer- und Gruppenkonti sowie von Ordnern+Dateien (inklusive aller Zugriffsrechte!) von Novell Netware NDS (Netware Directory Services) nach Windows 2000 / Active Directory. Riesige Zeitersparnis gegenüber manuellem Vorgehen, macht den Prozess überhaupt erst möglich.

? Wurde der Nutzen quantifiziert? Wenn ja, in welcher Form? Wenn nein, warum nicht?
Ja, in Form von Zeitvergleich (und damit Kostenvergleich) – mit / ohne Produkt.

Wie wurde das Projekt abgewickelt?

? Nach welcher Methode wurde vorgegangen (z.B. Extreme Programming)?
Keine spezifische / besondere Methode – Anforderungen kommen von potentiellen Kunden via Consultants und Product Management in Prosa-Form; detaillierte Anforderungen werden mit einem Geflecht von Excel-Sheets sehr detailliert erfasst (in Zs.arbeit Product Management / Entwicklung / Quality Control); OO-basierter Design (meist nur mental, nur wenig schriftlich); Umsetzung in Delphi / SQL in einem Dreierteam – Lead Dev veranwortlich für Applikation Architektur, Projektvorgaben, Datenbank-Design etc.

? Wie wurden die Projektmitarbeiter auf die verwendete Methode vorbereitet?
Keine spezifisches Training / Vorbereitung – Design wird diskutiert (Design review) und verfeinert, jedes Team Mitglied implementiert spezifische Teile der Applikation und ist dafür gesamtverantwortlich.

? Wie wurde geprüft, ob die Methode richtig angewendet wurde?
Design Reviews und später Code Review (Peer Reviews – Design wurde im ganzen Team diskutiert, Code unter den Entwickler/innen).

? Wurden spezielle Werkzeuge verwendet?
Nein.

Grundsatz 1: Einbezug Benutzer

? Wurde dieser Grundsatz eingehalten? *4*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?
Kein "Kundenprojekt", sondern "shrink wrap software" – Einbezug der potentiellen Kunden durch Feedback unserer Consultants draussen bei den Kunden. Gewisse Consultants wurden enger eingebunden, und hatten auch Kunden, die im Beta-Programm mitmachten.

? Wer hat die Spezifikation geschrieben? *Product Management*

? In welcher Form wurde die Spezifikation erstellt?
Word Dokument beschreibt Spezifikation aus Sicht des Product Managements / Marketings. Wurde in eine detaillierte Excel-Tabelle umgesetzt, die auch als Grundlage für die Entwicklung und Tests diente.

? Wurde die Spezifikation im Verlaufe des Projektes angepasst und wenn ja, warum?
Ja, Feedback von aussen, neue Feature Requests, Änderungen nach Erfahrungen mit der Vorläufer-Version; Usability Tests.

? Wann erhielten die Anwender erste Programmversionen? Anteil Projektgesamtdauer *80%*
Beta-Version war ca. Ende Juli (bei Entwicklung Oct 2000 – Januar 2001 / April – 13. Sept 2001) verfügbar

? Wie häufig erhielten die Anwender neue Versionen?
1 Beta-Version und 1 Release Candidate vor dem offiziellen Release.

? Wie wurde überprüft, ob die Spezifikation eingehalten wurde?
Rigorose Quality Control durch 2-3 Personen inhouse, Feedback von Beta-Programm.

Ich finde den Grundsatz "Einbezug Benutzer"
wichtig für das Gelingen eines Projektes: *6*

? Können Sie Ihre Gewichtung begründen ?
Der Benutzer sollte möglichst rasch etwas sehen, etwas einsetzen können – erst der tagtägliche Einsatz offenbart wirklich, ob die Bedürfnisse und Anforderungen abgedeckt sind. Das ist bei Kundenprojekten wesentlich einfacher, als bei "shrink wrap software"

Grundsatz 2: Einfach und klein

? Wurde dieser Grundsatz eingehalten? *4*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen ?
Kundenanforderungen, teilweise auch Druck von Seiten der Konkurrenz (Feature-Vergleiche)

? Enthält das Programm Funktionen, die nie gebraucht werden?
Ja.

? Falls unötige Funktionalitäten vorhanden sind, warum?
Feature-Druck von der Konkurrenz – wenn sie es haben, müssen wir es auch haben, sonst wird uns das ständig als Manko vorgeworfen, obwohl es oft total unnötig ist…..

? Wieviel Schulung ist nötig, um mit dem Produkt zu arbeiten?
In der Regel werden die Kunden von einem Consultant während mindestens 5 Tagen direkt am Produkt, direkt im Projekt geschult – Grossteil der Zeit wird für das Erstellen des Migrationsplans benötigt, direkte Programm-Schulung ist minimal – max. 1 Tag pro Kunde und Projekt.

? Wie wurde die Arbeit unter den Projektmitarbeitern aufgeteilt?
Nach Zeit / Verfügbarkeit, technischen Kenntnissen (Schwierigkeitsgraden), persönlichen Vorlieben (eher User Interface orientiert, oder eher nicht), nach Bedarf von Seiten des Projekts und aufgrund Einschätzung des Lead Developers.

? Was waren die Gründe für die gewählte Aufteilung?
Primär Kenntnisse, Neigungen und Bedarf.

? Führte die gewählte Aufteilung zu Problemen? Wenn ja, welche?
Keine wesentlichen bisher.

? Wieviele Projektmitglieder kennen den gesamten Sourcecode?
Alle – in unterschiedlichem Ausmass. Wirklich zu 100%: der Lead Developer alleine. (wenn überhaupt jemand)

Ich finde den Grundsatz "Einfach und klein"
wichtig für das Gelingen eines Projektes: *5*

? Können Sie Ihre Gewichtung begründen ?
Recht wichtig, allerdings nicht das wichtigste – je nach Aufgabe des Projekts kann es relativ weitreichend und damit eher gross und schwerfällig sein.

? Haben Sie weitere Bemerkungen zum Grundsatz "Einfach und klein"?
Es ist eine alte Weisheit der Software Entwicklung, dass die "do-it-all" Anwendungen in der Regel alles ein wenig, aber fast nichts wirklicht gut machen. Trend geht dahin, lieber kleinere Entitäten zu haben, die eine spezifische Aufgabe sauber und gut lösen, und mit anderen Entitäten zusammenarbeiten können.

Grundsatz 3: Qualität in allen Teilen

? Wurde dieser Grundsatz eingehalten? *6*

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?
Sehr wenige Fehlermeldungen von den Consultants draussen bei den Kunden; sehr wenige Anrufe beim Support (im Vergleich zu anderen Produkten aus dem gleichen Hause / in ähnlichen Aufgabengebieten).

? Gibt es im Endprodukt bekannte, nicht behobene Fehler?
Ja.

? Warum sind die Fehler vorhanden?
Fehler passieren eben – Schreibfehler, Fehler beim Verständnis der Anforderungen, Logikfehler beim Entwickeln der Applikation.

? Warum werden die Fehler nicht behoben?
Sie wurden teils erst sehr spät im Testverfahren entdeckt, und als zu wenig dringlich eingestuft, als dass man eine Verzögerung des Termins und / oder eine Beeinträchtigung der Gesamtqualität des Produkts riskieren wollte. Grösstenteils kleine, ärgerliche Punkte, teils z.B. Schreibfehler auf Dialogboxen etc.

? Wurden regelmässig Tests durchgeführt?
Ja, ca. 8 Wochen vor dem Release-Datum war RTQC (Release To Quality Control).

? In welcher Form wurde getestet?
Basierend auf der sehr detaillierten Liste der Anforderungen wurden Tests entwickelt und das erwartete Ergebnis festgehalten. Diese Tests wurden anschliessend (teils manuell, teils mit automatisierten Test Tools) durchgeführt und die Ergebnisse mit den Erwartungen verglichen, und Abweichungen in einem Quality Control Software Paket als Bugs vermerkt.

? Gibt es Teile in der Software die neu geschrieben werden sollten?
Ja.

? Warum sollten welche Teile neu geschrieben werden?
Steigerung der Performance.

? Gab es eine institutionalisierte Form für Design- und Codereview?
Ja.

? In welcher Form wurden die Reviews durchgeführt?
Peer Reviews – Designreviews wurden im gesamten Team 12x wöchentlich durchgeführt – Designer stellt Ideen vor, diese werden diskutiert und bewertet und allenfalls geändert. Code Reviews: 1x wöchentlich unter den Entwicklern.

? Ist die Software hinreichend dokumentiert?
Nein.

? Welche Art von Dokumentationen wurden erstellt?
Kommentare im Source Code, teilweise Design Dokumente.

Ich finde den Grundsatz "Qualität in allen Teilen"
wichtig für das Gelingen eines Projektes: *6*

? Können Sie Ihre Gewichtung begründen ?
Software-Qualität ist absolut vordringlich – lieber gute Software später, als ein Gebastel heute.

? Haben Sie weitere Bemerkungen zum Grundsatz "Qualität in allen Teilen"?
Lässt sich leider oftmals nicht mit Zeitvorgaben / Marktdruck vereinbaren.

Grundsatz 4: Zeitplan

? Wurde dieser Grundsatz eingehalten? *4*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?
Es wurden vordringlichere, direkter ertragswirksame Klein-Projekte vorgezogen und sie hatten eine klar negative Auswirkung auf den Zeitplan des Hauptprojekts. Bewusster Business Entscheid in vollem Bewusstsein der Verzögerungen.

? Wie wurde der Zeitplan festgelegt? *schriftlich*

? Wie detailliert war der Zeitplan aufgeteilt?
Pro Person bis auf Tagesebene.

? Waren jedem Mitarbeiter der Zeitplan / Ziele bekannt?
Ja.

? Wie wurde die Einhaltung Ziele / Zeitplan überprüft?
Wöchentliche Teamsitzungen.

? Wie wurden auf Abweichungen Realität <-> Zeitplan reagiert?
Neu-Bewertung der Prioritäten – Abschätzung der Folgen einer Verzögerung vs. Folgen des Streichens von Features.

Ich finde den Grundsatz "Zeitplan"
wichtig für das Gelingen eines Projektes: *5*

? Können Sie Ihre Gewichtung begründen ?
Der Zeitplan ist wichtig, aber primär als Hilfsmittel für die Betroffenen, um zu wissen, wo man steht, wie man im Zeitplan liegt, und wann in etwa das Projekt wirklich fertig sein wird.

? Haben Sie weitere Bemerkungen zum Grundsatz "Zeitplan"?
Zeitplan sollte nicht vor Qualität kommen, wenn immer möglich – Einhaltung des Zeitplans heisst fast immer, dass man bei den Features, der Qualität, oder beiden, Abstriche machen muss.

Grundsatz 5: Spass der Beteiligten

? Wurde dieser Grundsatz eingehalten? *5*

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?
Nahezu 0% Turnover im Projektteam.

? Was war Ihre wichtigste Motivation bei der Arbeit an diesem Projekt? Warum?
Erreichung der selber gesteckten Ziele (in Hinsicht auf Projektumfang, und Zeitrahmen).

? Gab es für sie persönlich wichtige Ziele, die nicht erreicht wurden? Welche? Warum?
Nein.

? Was hat Ihnen bei diesem Projekt am wenigsten gefallen? Warum?
Erste Runde der Anforderungen war bei weitem zu weit, zu umfangreich – das Projekt geriet in eine Schieflage, weil zuviel aufs Mal realisiert werden sollte.

? Was hat bei diesem Projekt besonders gut funktioniert? Warum?
Teamwork war hervorragend, Product Management, Projektmanagement, Entwicklung, Support, Quality Control, Tech Writing, alle haben super zusammengearbeitet.

? Wenn Sie rückblickend Ihre Rolle im Projekt ändern könnten, was würden Sie ändern?
Nichts.

? Gab es im Laufe des Projektes Konflikte unter den beteiligten Personen?
Ja, sicher.

? Wie wurden aufgetretene Konflikte gelöst?
Durch Gespräche, Teamsitzungen, Klärung, was genau gemeint war etc.

? Gab es personelle Wechsel im Projektteam? Gründe ?
Ja – zwei Abgänge von "peripheren" Mitarbeitern (die nur zum Teil für unser Projekt arbeiteten). Persönliche Gründe für einen Wechsel, beide nach etlichen Jahren auf der Suche nach neuen Herausforderungen.

? Gab es gesellschaftliche Anlässe innerhalb des Projektteams? Welche? Von wem initiert?
Ja, ab und zu ein Mittagessen, Release Party am Ende. Meist organisiert vom Projektleiter, auf Anregung von anderen Teammitgliedern, aber auch spontan unter den Teammitgliedern.

? Nahmen Sie an solchen Anlässen teil? Warum? Warum nicht?
Ja – fördert das persönliche Verhältnis, stärkt Teambildung, Teamzusammenhalt.

Ich finde den Grundsatz "Spass der Beteiligten"
wichtig für das Gelingen eines Projektes: *6*

? Können Sie Ihre Gewichtung begründen ?
Es ist von eminenter Wichtigkeit, dass die Teammitglieder Freude an der Arbeit haben – nur so können sie genügend motiviert sein, Qualität zu erbringen und Termine einzuhalten. Software-Entwicklung ist weitgehend ein kreativer Prozess, da muss die Stimmung i.O. sein, sonst geht nichts voran.

Fragebogen / Interview Projekt 2

Allgemeine Angaben zum Projekt

Ihre Rolle im Projekt:	*A: Anwender B, D: Entwickler C: Projektleiter*		
Fachgebiet:	Business Software		
Zeitdauer des Projektes:	Beginn am:..... *1996*	Ende am: ..... *noch laufend*	
Anzahl beteiligte Personen:	Total: *5*	davon Entwickler: *2*	
Aufwand in Arbeitstagen:	Total: .....*425 Tage*	davon Entwicklung:..... *280 Tage*	
Umfang des Projektes:	Anzahl Applikationen: *5*		
Anzahl Zeilen Sourcecode:	dieses Projekt: *35K*.	eigene Library: *50K*..	Drittlibraries: *> 200K*
Anzahl Klassen:	dieses Projekt: *100*.	eigene Library: *120*.	Drittlibraries: *> 300*
Programmiersprache(n):	*Object Pascal*		
Entwicklungsumgebung(en):	*Delphi, MS SQL Server*		

War das Projekt erfolgreich?

Termine wurden eingehalten:	*4: A 6: B, C, D*
Kostenrahmen wurde eingehalten:	*5: A 6: B, D 7: C*
Software ist fehlerfrei:	*3: A 5: B, D 6: C*
Software erfüllt die Benutzeranforderungen:	*6: A, C, D 7: B*
Software ist wartbar:	4: D *5: A, B 6: C*

? Welche Rahmenbedingungen machten das Projekt speziell einfach?

A: Benutzer wusste genau was er will. Zudem waren nur wenige Leute beteiligt und die Zusammenarbeit mit dem Kunden war sehr gut.

B: Termine waren relativ offen. Kontakt mit Kunde war immer sehr gut. Budget war fix und wir bestimmten, welche Arbeiten dafür möglich sind. Optimale Bedingungen für uns als Softwarehersteller.

C: Das Knowhow zu Verkehrsfragen war bei uns vorhanden. So haben wir in wesentlichen Teilen auch die Rolle des Auftraggebers übernehmen können und vorgeschlagen, welche Funktionalitäten nötig und wichtig sind.

D: Die verwendete Entwicklungsumgebung (Delphi).

? Welche Rahmenbedingungen machten das Projekt speziell kompliziert?

A: Der Hauptentwickler wechselte und der neue Entwickler wusste nur sehr wenig vom Fachgebiet "Verkehrszählungen".

B: Kommunikation der einzelnen Applikationen über Modem. Einzelne Applikationen laufen an verschiedenen Orten, teilweise halb draussen.

C: Vor allem die Hardwareprobleme mit Standleitungen, Detektoren sowie die verteilte Installation an mehreren Standorten. Kompliziert war es auch, weil der Auftraggeber selber keinen Projektleiter gestellt hat, der die Bedürfnisse formuliert und diese auch aktiv überwachen würde.

D: Probleme mit Hardware (Modems, Detektoren).

? Was ist der Nutzen der erstellten Software?

A: Nutzen ist sehr gut. Software erfüllt genau die Anforderungen des Kunden.

B: Software erfüllt Kundenanforderungen. Ob es wirklich benutzt wird, kann ich nicht beurteilen.

C: Zentrale statistisches Daten über Verkehrsmengen, automatisierte Auswertungen. Wegen Arbeitsüberlastung beim Kunden, wird die Software eigentlich zuwenig genutzt.

? Wurde der Nutzen quantifiziert? Wenn ja, in welcher Form? Wenn nein, warum nicht?

A: Weiss ich nicht.

B: Nein, vermutlich nicht.

C: Im Kreditantrage wurde das sicher quantifiziert.

Wie wurde das Projekt abgewickelt?

? Nach welcher Methode wurde vorgegangen (z.B. Extreme Programming)?

B: Keine Methode.

C: In der zweiten Runde wurde versucht, die klassische Phasenmethode anzuwenden. Obwohl es im wesentlichen nur ein Umschreiben der bestehenden Software war zeigte sich, dass einfach nicht alles im voraus planbar ist.

D: Weiss ich nicht.

? Wurden spezielle Werkzeuge verwendet?

B: Designtool für Datenbanken. Hat sich bewährt.

D: Tool für Datenbankdesign.

Grundsatz 1: Einbezug Benutzer

? Wurde dieser Grundsatz eingehalten? *6: A, C, D 7: B*

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Sowohl Testbenutzer bei uns als auch Benutzer beim Auftraggeber waren während der ganzen Projektabwicklung integriert.

B: Der Kontakt mit dem Kunden war sehr gut. Besprechungen waren jederzeit möglich und das gegenseitige Verständnis war da. Das Produkt macht, was der Kunde will.

C: Da wir das Fachwissen im Haus hatten und eigentlich die Rolle des Benutzers selber übernehmen konnten, war dieser Grundsatz sicher eingehalten.

D: Ich hatte wenig direkten Kontakt mit Endbenutzer. Diese Kontakte gingen praktisch alle über den Projektleiter. Deshalb ist es nicht ganz einfach zu beurteilen, ob dieser Grundsatz eingehalten wurde. Trotzdem war der Kontakt ziemlich eng zum Benutzer, einfach über den Projektleiter.

? Wer hat die Spezifikation geschrieben?

A: Weiss nicht.

B: Projektleiter und Entwickler, vom Kunden nur mündlich.

C: Projektleiter.

D: Projektleiter.

? In welcher Form wurde die Spezifikation erstellt?

A: Weiss nicht.

B: Konzepte als Worddokumente. Wurden leider im Laufe des Projektes nicht aktualisiert.

C: Schriftlich, mindestens teilweise. Für die erste Version wurde ein Pflichtenheft erstellt. Es gab zwei Arten von Erweiterungen: Kleine Änderungen oder Korrekturen wurden mit Notizen oder mündlich kommuniziert, für neue Module gab es Beschreibungen in schriftlicher Form.

D: Schriftlich und mündlich, keine besondere Form.

? Wurde die Spezifikation im Verlaufe des Projektes angepasst und wenn ja, warum?

A: Ja, weil die Anforderungen vom Kunden her gewechselt haben, die bei Projektbeginn nicht bekannt waren. Da es sich um ein Dauerprojekt handelt kommen jedes Jahr neue Anforderungen dazu.

C: Ja es gab Anpassungen, welche aber nie in einem Gesamtdokument nachgeführt wurden.

D: Ja, laufend. Wird immer noch angepasst.

? Wann erhielten die Anwender erste Programmversionen?

A: Weiss nicht.

B: Schwierig zu sagen. Jeweils im Jahreszyklus kamen Anforderungen dazu, welche als Ganzes implementiert und getestet wurden. So gesehen war jedes Jahr ein Teilprojekt.

C: Laufende Tests intern der fertigen Module.

? Wie häufig erhielten die Anwender neue Versionen?

A: Neue Versionen fast täglich, allerdings nur für den Testanwender bei uns intern. Die Anwender beim Kunden erhielten weniger häufig neue Versionen

B: Jährlich, abgesehen von einzelnen Bugfixes.

D: Etwa im Jahresrythmus.

? Wie wurde überprüft, ob die Spezifikation eingehalten wurde?

A: Vermutlich gar nicht. Der Endanwender beim Kunden hat vermutlich schon überprüft, ob alle Funktionalitäten, die abgemacht wurden, vorhanden sind und so kann man davon ausgehen, dass die Spezifikation erfüllt ist.

B: Im Einsatz, es gab keine eigentliche Abnahme.

C: Durch interne Tests. Zweimal gab es eine offizielle Abnahme beim Auftraggeber.

D: Nur wir selber in der Entwicklerfirma. Auch mit der Installation vor Ort hatte der Kunde nichts zu tun.

Ich finde den Grundsatz "Einbezug Benutzer"
wichtig für das Gelingen eines Projektes: *7: A, B, C, D*

? Können Sie Ihre Gewichtung begründen ?

A: Absolut nötig. Sonst läuft ein Projekt in die falsche Richtung.

B: Projekte, wo Entwickler sich selber verwirklichen führt häufig zu Problemen.

C: Programmierer haben meistens das fachliche Know-How nicht, um beurteilen zu können, ob ihre Arbeit den Benutzeranforderungen entspricht.

D: Gute Spezifikationen erleichtern die Programmierarbeit, also muss der Benutzer mitarbeiten.

? Haben Sie weitere Bemerkungen zum Grundsatz "Einbezug Benutzer"?

C: Es funktioniert gut, wenn der Benutzer im Projekt integriert sind. Häufig nehmen sich aber Benutzer zu wenig Zeit, um sich mit dem Projekt auseinanderzusetzen.

D: Ein immer wieder auftauchendes Problem ist, dass der Benutzer nicht weiss was er will.

Grundsatz 2: Einfach und klein

? Wurde dieser Grundsatz eingehalten? *2: D 4: B 6: C*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Ist für mich schwierig zu beurteilen.

B: Bei einem Projekt, das sich über Jahre dahinzieht ist das nicht immer einfach einzuhalten. So gesehen ist es heute nicht mehr "einfach und klein".

D: Das ganze System ist zu monolythisch. Es ist schon aufgeteilt in Teilapplikationen, aber einzelne davon sind zu lange einfach gewachsen. Wenn man es neu schreiben würde, würde das ganze viel schlanker.

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

C: Das Gesamtprojekt wurde in einzelne Applikationen aufgeteilt, welche als Teilprojekte einzeln abgeschlossen werden konnten.

? Enthält das Programm Funktionen, die nie gebraucht werden?

A: Nein, es gibt keine.

B: Nein, keine. Diese wurden alle immer wieder entfernt.

C: Nein.

D: Garantiert.

? Falls unötige Funktionalitäten vorhanden sind, warum?

D: Kann es nicht genau sagen. Vermutlich wurden Funktionalitäten einmal spezifiziert, aber ich denke gewisse Teile werden nie verwendet.

? Wieviel Schulung ist nötig, um mit dem Produkt zu arbeiten?

A: Wenig Schulung nötig. Die Fachpersonen kennen die verwendeten Begriffe. Die Software ist weitgehend selbsterklärend.

B: Es gibt zwei Seiten. Das Mangementtool für den Kunden ist sehr einfach in der Anwendung. Schwieriger sind die Applikationen, welche verteilt zur Datenerfassung laufen.

C: Im Prinzip nicht nötig. Die Auswertungen sind selbsterklärend. Schulung würde erst nötig, wenn andere Ämter damit arbeiten würden, wobei es sich hier vor allem um eine fachtechnische Schulung handeln würde.

D: Schwierig zu sagen, da viele Arbeiten nur bei uns intern erledigt werden. Zwei bis drei Tage sollten reichen.

? Wie wurde die Arbeit unter den Projektmitarbeitern aufgeteilt?

A: Jeder machte das was er kann (Entwicklung - Tests/Dokumentation - Projektleitung).

B: Aufteilung ergab sich aus der Ausbildung/Können der Projektmitglieder. Projektleiter übernahm schwergewichtig die Kundenkontakte.

C: Gegeben durch Ausbildung der Leute.

D: Gegeben durch Ausbildung der Leute.

? Was waren die Gründe für die gewählte Aufteilung?

A: Gegeben durch Wissen und Können der beteiligten Personen.

? Führte die gewählte Aufteilung zu Problemen? Wenn ja, welche?

A: Es gab keine Probleme.

C: Höchstens wegen der leicht unterschiedlichen Arbeitsweise der beiden involvierten Entwickler.

D: Nein.

? Wieviele Projektmitglieder kennen den gesamten Sourcecode?

A: Vermutlich niemand.

B: Im Moment noch ich, sonst vermutlich niemand.

C: Ich nehme an, zwei Personen.

D: Niemand. Ich kenne allenfalls etwa die Hälfte.

Ich finde den Grundsatz "Einfach und klein"
wichtig für das Gelingen eines Projektes: *5: A 6: C 7: A, D*

? Können Sie Ihre Gewichtung begründen ?

C: Einfach und klein ist immer wichtig, kann aber nicht bei jedem Projekt eingehalten werden.

D: Ist nicht immer einfach und möglich, diesen Grundsatz einzuhalten, aber ein Projekt bleibt besser wartbar.

? Haben Sie weitere Bemerkungen zum Grundsatz "Einfach und klein"?

B: Grundsatz ist wichtig, wie gut er einzuhalten ist, ist schwierig. Um "einfach und klein" zu bleiben ist vermutlich so alle fünf Jahre ein Redesign nötig. In einem laufenden Projekt ist "einfach und klein" irgendwann nicht mehr möglich. Solche Programme bergen auch die Gefahr, sehr heterogen zu werden.

Grundsatz 3: Qualität in allen Teilen

? Wurde dieser Grundsatz eingehalten? *3: D 4: C 6: A, B*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Am Schluss gab es sehr häufig neue Versionen, manchmal auch mit neuen Fehlern.

C: Es fehlte in den Anfängen eine saubere Planung. Das ganze ist mehr organisch gewachsen. Vermutlich hat es auch in der aktuellen Version noch "Altlasten" von früher.

D: Etwas, dass ich in diesem Projekt vermisst habe sind Testsequenzen für gewisse Funktionalitäten.

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

B: Ist aus meiner Sicht eine Grundvoraussetzung für gute Projekte. Qualität ist nicht ein "Nice to have" sondern ein "Need to have". Dies wird nicht in allen Firmen so gehandhabt. Was bei diesem Projekt zu kurz kam, waren Grundkonzepte am Anfang. Da wäre sicher noch etwas herauszuholen.

? Gibt es im Endprodukt bekannte, nicht behobene Fehler?

A: Es gibt Fehler.

B: Meines Wissens keine.

C: Nur unbekannte.

D: Nur kleinste Fehler.

? Warum werden die Fehler nicht behoben?

A: Aufwand zur Behebung wäre zu gross und die Software ist auch mit diesen Fehlern verwendbar.

D: Fehler ist klein und verursacht keine akuten Probleme.

? Wurden regelmässig Tests durchgeführt?

A: Ja durch Testanwender, bei Entwicklern vermutlich nur teilweise.

B: Wir haben eine kleine Testumgebung mit Detektoren hier im Büro, aber gewisse Tests waren nur im laufenden Betrieb möglich, da nicht ganz alles simuliert werden kann.

C: Ja, Funktionstest der Module.

D: Ja, im laufenden Betrieb.

? In welcher Form wurde getestet?

A: Fanden "on the job" statt, dass heisst mehrheitlich direkt mit den produktiven Daten. Deshalb mussten gewisse Batchjobs mehrfach durchgeführt werden.

B: Keine formalen Tests mit Vorgaben und Protokollen. Ist sicher ein Punkt, der noch verbessert werden könnte.

C: Während dem produktiven Einsatz. Hat funktioniert, weil wir bei den kritischen Funktionen selber Benutzer waren. Automatische Tests wären gut, aber die Probleme traten immer bei irgendwelchen Spezialfällen auf, an die vorher niemand gedacht hatte.

? Gibt es Teile in der Software die neu geschrieben werden sollten?

A: Aus Anwendersicht keine.

B: Das Projekt ist schon so alt, dass es sicher Teile gibt, die neu geschrieben werden sollten, schon nur weil man heute für Teilprobleme einfachere Lösungen kennt. Aber das System funktioniert und läuft.

C: Der aktuelle Prozess, wo die Daten plausibilisiert werden ist zuwenig transparent. Dieses Modul sollte sicher einmal neu konzeptioniert und programmiert werden.

D: Schwierig zu sagen. Falls das Programm als Produkt verwendet werden soll, werden sich vermutlich ein paar grundsätzliche Probleme stellen (zuviele verschiedene Units).

? Warum sollten welche Teile neu geschrieben werden?

D: Für jede Art von Analyse und Transfers wurden verschiedene Datentypen definiert. Meiner Meinung nach zuviel verschiedene, aber dass ist auch eine Frage des Programmierstils der einzelnen Programmierer. Auch die Kommunikation würde ich heute mit bekannten und gebräuchlichen Internetprotokollen lösen und nicht selber ausprogrammieren.

? Gab es eine institutionalisierte Form für Design- und Codereview?

A: Weiss nicht.

B: Gab es nicht.

C: Nein. Es wurde nach einfachen internen Richtlinien entwickelt. Jeder Entwickler für seinen Code verantwortlich.

? In welcher Form wurden die Reviews durchgeführt?

C: Eigentlich keine. Code und Designreviews wären allenfalls sinnvoll, müssten dann aber von Anfang anlaufend stattfinden. Zudem muss die nötige Zeit dafür bereitgestellt werden.

? Ist die Software hinreichend dokumentiert?

A: Mittlerweile ja.

B: Nein. Ich denke für jemand, der gerne etwas liest, ist zuwenig Dokumentation vorhanden. Für die aktuellen Mitarbeiter reicht es, da diese vor allem den Sourcecode lesen.

C: Handbücher sind gut. Installationsanleitungen sind vorhanden. Eine einfache Systembeschreibung ist vorhanden, ob die genügend ist kann ich nicht beurteilen.

D: Ja und nein. Ein paar zusätzliche Klassendiagramme und Beschreibungen der Kommunikationsflüsse wäre praktisch, aber das meiste ist sowieso besser im Sourcecode ersichtlich.

? Welche Art von Dokumentationen wurden erstellt?

A: Technische Dokumentation und Benutzerhandbücher für alle Einzelprogramme.

B: Ein paar Seiten Systemübersicht und ein Datenbankmodell.

Ich finde den Grundsatz "Qualität in allen Teilen"
wichtig für das Gelingen eines Projektes: *7: A, B, C, D*

? Können Sie Ihre Gewichtung begründen ?

C: Wie bei allen Qualitätsansprüchen gilt auch hier, dass Qualität nur erreicht wird, wenn man vom Anfang an darauf hin arbeitet.

? Haben Sie weitere Bemerkungen zum Grundsatz "Qualität in allen Teilen"?

B: Eine Mindestqualität ist nötig. Der Programmcode ist aus meiner Sicht schlussendlich das wichtigste. Es ist durchaus möglich, gute Projekte zu realisieren, die eine mangelhafte Planung und Dokumentation aufweisen. Kundenzufriedenheit ist schlussendlich das wichtigste Qualitätsmerkmal.

D: Ist immer eine Frage, ob die nötige Zeit dazu vorhanden ist.

Grundsatz 4: Zeitplan und Ziele

? Wurde dieser Grundsatz eingehalten? 3:B *5: A 6: C, D*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Nicht absichtlich, neue Features waren gemacht, dafür waren manchmal andere Fehler vorhanden.

C: Die Konvertierung von Version eins zu zwei war zuwenig genau geplant und der Aufwand war schlussendlich grösser als angenommen.

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Endtermine beim Kunden wurden eingehalten, interne Zwischentermine wurden nicht immer eingehalten, was die Arbeit manchmal erschwerte.

B: Die Termine waren sehr offen. Es gab keinen eng abgesteckten Zeitplan. Der jährliche Termin konnte auch um einen Monat verschoben werden, da der Kunde sehr flexibel war.

C: Für die einzelnen Jahre waren die Ziele für neue Module recht klar definiert.

D: Ich denke, die Termine wurden immer eingehalten.

? Wie wurde der Zeitplan festgelegt?

A: Sicher nicht schriftlich, vermutlich mündlich oder gar nicht.

B: Mündlich

C: Schriftlich, für einzelne neue Module. Umstellung Datenbank (Version 1->2) nur mündlich.

D: Weiss ich nicht genau.

? Wie detailliert war der Zeitplan aufgeteilt?

A: Nur Endtermin

B: Nur Endtermin (jährlich). Es gab keine Teilziele

C: Es gab eine Aufwandschätzung und einen Endtermin.

D: Höchstens so im Jahresrythmus, gröbere Fehler sofort.

? Waren jedem Mitarbeiter der Zeitplan / Ziele bekannt?

A: Ja, war sowohl den Mitarbeitern als dem Kunden bekannt (Endtermin).

B: Ja, war auch nicht ein Problem, da der Zeitplan sehr einfach war.

C: Ja.

D: Nein.

? Wie wurde die Einhaltung Ziele / Zeitplan überprüft?

A: Ja und nein. Der Kunde verliess sich auf unsere Zusicherung, dass die Arbeiten erledigt sind.

? Wie wurden auf Abweichungen Realität <-> Zeitplan reagiert?

A: Es war keine Reaktion nötig, da der Endtermin beim Kunden eingehalten werden konnte.

B: War nie nötig, da die Flexibilität mit den jährlichen Terminen.

C: Umstellung Version eins nach zwei brauchte mehr Zeit. Da mit der Version 1 weitergearbeitet werden konnte, war es nicht ein Problem, den Endtermin zu verschieben.

Ich finde den Grundsatz "Zeitplan"
wichtig für das Gelingen eines Projektes: *4: C 5: B, D 6: A*

? Können Sie Ihre Gewichtung begründen ?

A: Es kann sein, dass ein Zeitplan nicht eingehalten werden kann. Qualität und Fehlerfreiheit sollte auf alle Fälle stärker gewichtet werden.

C: Häufig liegt es nicht bei uns als Auftragnehmer, sondern beim Auftraggeber, welche bei seine Beiträge zum Projekt (Stellungnahme, Tests) nicht termingerecht liefert. Weiter sind Zeitschätzungen von Entwicklern nicht immer gleich zuverlässig.

D: Kommt sehr auf das Projekt an. Soll das Endprodukt qualitativ gut sein oder muss ein bestimmter Termin unbedingt eingehalten werden. Diese zwei Teile sind gegeneinander abzuwägen.

? Haben Sie weitere Bemerkungen zum Grundsatz "Zeitplan"?

B: Bei grossen Projekten sind Zwischenziele sicher unumgänglich. Herunterbrechen auf Tage ist sicher nicht nötig, eher Meilensteine für gewisse Programmteile. Je grösser das Team, desto wichtiger wird sicher auch der Zeitplan, da mehr auf andere Leute Rücksicht genommen werden muss.

D: Zu detaillierte Planung (tagesweise) ist bei der Informatik eine Illusion, da fast immer irgend etwas unvorhergesehenes auftreten kann.

Grundsatz 5: Spass der Beteiligten

? Wurde dieser Grundsatz eingehalten? *3: B 4: A, D 6: C*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Die Abschlussphase war sehr mühsam.

B: Ist gegeben durch das Projekt. Zulange am selben Projekt arbeiten macht irgendwann keinen Spass mehr. Besonders mühsam war auch die Zusammenarbeit mit Drittfirmen, wie hier z.B. mit der Swisscom, die wochenlang brauchte, um die nötigen Standleitungen sauber aufzuschalten.

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

D: Allgemein: Arbeitsklima ist gut, Das verwendete Werkzeug (Delphi) ist sehr gut. Man hatte viele Freiheiten bei der Arbeit. Bezogen auf das Projekt kann ich nur sagen, es ist nicht "mein Kind" und so hält sich die Motivation im Rahmen.

? Was war Ihre wichtigste Motivation bei der Arbeit an diesem Projekt? Warum?

A: Die Arbeit wurde mir so zugeteilt. Ich hatte keine Wahl.

B: Die Kommunikation zwischen den einzelnen Applikationen war die grösste Herausforderung. Aber irgendwann, sank die Motivation. Ich war zu lange mit Applikationen zum Thema Verkehrszählungen beschäftigt.

C: Verkehrszählungen ein wichtiger Bestandteil der Entstehungsgeschichte unserer Firma. Es war das erste grössere Softwareprojekt, dass bei uns abgewickelt wurde. Als Projektleiter sieht man die Einhaltung des Budgets auch immer als wichtiges Ziel.

D: Lerneffekt, da viele verschiedene Dinge vorkommen (Datenbanken, GUI, Grafik, Kommunikation).

? Gab es für sie persönlich wichtige Ziele, die nicht erreicht wurden? Welche? Warum?

A, D: Keine.

B: Keine, ausser vielleicht dass es nicht bis am Schluss Projekt "einfach und klein" blieb.

? Was hat Ihnen bei diesem Projekt am wenigsten gefallen? Warum?

A: Die vielen kleine Fehler bei der Entwicklung in der Abschlussphase: Zwei Schritte vorwärts, ein Schritt zurück.

B: Die Zusammenarbeit mit externen Firmen war mühsam.

C: Nichts.

D: Grösse, das ganze ist ein wenig unübersichtlich.

? Was hat bei diesem Projekt besonders gut funktioniert? Warum?

A: Die Probleme in der Abschlussphase waren für den Kunden nicht sichtbar.

B: Die unkomplizierte Zusammenarbeit mit Kunden und dem Projektleiter.

C: Die einzelnen Module passten schlussendlich wirklich zusammen.

D: Kommunikation zwischen den Mitarbeitern, vor allem auch bei der Einarbeitung.

? Wenn Sie rückblickend Ihre Rolle im Projekt ändern könnten, was würden Sie ändern?

A: Ich kann nichts ändern.

B: Nichts. Mir gefällt die Rolle als Umsetzer besser als die des Projektleiters, welcher stark mit Sitzungen und Administration belastet ist.

C: Eine Änderung wäre nur möglich, wenn von Seite Auftraggeber die Rolle des Benutzers besser wahrgenommen worden wäre. Dann hätte ich diese Aufgabe abgeben können.

D: Nichts.

? Gab es im Laufe des Projektes Konflikte unter den beteiligten Personen?

A: Ja, manchmal sprachen nicht alle vom Gleichen (bei Fehlerbehebung und Bereinigungen). Terminologie war manchmal auch unklar.

B: Nein, keine wirkliche Konflikte.

C: Nein. Es gab kleinere Diskussionen über die Vorgehensweise, aber das waren Details.

D: Ja, wegen Verständnisproblemen bei Anpassungen gewisser Funktionalitäten.

? Wie wurden aufgetretene Konflikte gelöst?

D: Im Gespräch, zusammen mit dem Projektleiter

? Gab es personelle Wechsel im Projektteam? Gründe?

A: Ja, Abgang des Hauptentwicklers, wobei das nicht im Zusammenhang mit diesem Projekt stand, war aber für das Projekt ein grosser Einschnitt.

B: Nein, ausser temporäre Aushilfe von anderen Entwicklern. Auf Kundenseite gab es ein Wechsel, aber auch der war unproblematisch.

C: Mehrere auf Seite des Auftraggebers.

D: Ja, aber nicht direkt im Zusammenhang mit diesem Projekt

? Gab es gesellschaftliche Anlässe innerhalb des Projektteams? Welche? Von wem initiert?

A: Ja schon, das grosse Abschlussessen ist allerdings noch ausstehend.

B: Keine.

C: Ein oder zwei Mittagessen mit Auftraggeber.

D: Projektspezifisch keine, aber sonst Anlässe innerhalb des

? Nahmen Sie an solchen Anlässen teil? Warum? Warum nicht?

B: Ja. Wenn man etwas erreicht hat, soll man es feiern und nicht einfach so das nächste Projekt starten.

Ich finde den Grundsatz "Spass der Beteiligten"
wichtig für das Gelingen eines Projektes: *6: C 7: A, D*

? Können Sie Ihre Gewichtung begründen ?

C: Es gibt Dinge die machen sehr viel Spass. Irgendwann sind dann noch die mühsamen letzten fünf Prozent zu erledigen, welche keine neue Herausforderung mehr bedeuten, sondern nur noch Arbeit. Das sind zum Beispiel die Tests ob alle Funktionalitäten wirklich sauber funktionieren.

Aber genau diese letzten fünf Prozent garantieren, dass das Projekt schlussendlich sauber funktioniert. Deshalb Spass ist gut, aber die langweilige Arbeit muss auch erledigt werden.

? Haben Sie weitere Bemerkungen zum Grundsatz " Spass der Beteiligten "?

B: In einem Projekt, dass Spass macht, sind bessere Leistungen möglich. Mit einer professionellen Einstellung kann aber auch ein Projekt realisiert werden, dass nicht nur Spass macht. Häufig fehlt diese Einstellung. In jedem Projekt ist mal klar, wie alles realisiert wird und dann kommt noch die Fleissarbeit, die nicht mehr unbedingt interessant ist. Und diese Fleissarbeit sauber zu erledigen, gehört zu einer professionellen Einstellung.

D: Ich würde es nicht unbedingt Spass nennen. Spass und Arbeit sind nicht immer vereinbar. Ich würde eher von Klima und Umfeld sprechen.

Fragebogen / Interview Projekt 3

Allgemeine Angaben zum Projekt

Ihre Rolle im Projekt:	*A: Projektleiter/CIO B: Entwickler C: Leiter Entwicklung*
Fachgebiet:	*Internet Applikation*
Zeitdauer des Projektes:	Beginn am: *Mai 2000*..... Ende am: *Juli 2001*
Anzahl beteiligte Personen:	Total: *18*..... davon Entwickler: *14*
Aufwand in Arbeitstagen:	Total: *5500*..... davon Entwicklung: *4500*
Umfang des Projektes:	Anzahl Applikationen: *12*
Anzahl Zeilen Sourcecode:	dieses Projekt: *300K* eigene Library: ..... Drittlibraries:..... *?*
Anzahl Klassen:	dieses Projekt: 2'700 eigene Library: ..... Drittlibraries:..... *?*
Programmiersprache(n):	*Java / Javascript / HTML / Perl*
Entwicklungsumgebung(en):	*Forte for Java, iAS 6.0, LDAP, Oracle DB, iWS 4x*

War das Projekt erfolgreich?

Termine wurden eingehalten:	*4: B,* 6: *A, C*
Kostenrahmen wurde eingehalten:	*5: A, 6: C, 7: B*
Software ist fehlerfrei:	*4: A, 5: B, C*
Software erfüllt die Benutzeranforderungen:	*4: C, 6: B*
Software ist wartbar:	2: C, *4: A, B*

? Welche Rahmenbedingungen machten das Projekt speziell einfach?

A: Da es sich um eine Internetapplikation handelt, konnten Patches sehr einfach eingespielt werden.

B: Nice working environment, existence of extended libraries in Java.

C: Das technische Umfeld (Java) hat die Arbeit sicher vereinfacht.

? Welche Rahmenbedingungen machten das Projekt speziell kompliziert?

A: Es wurden viele neue Produkte verwendet, junge Technologie, die häufig noch fehlerhaft war. Leute suchten tagelang Fehler, die eigentlich in einem Fremdprodukt waren.

B: Absence of an actual customer, frequent scope changes, absence of clear technical specifications and development methodology at the beginning, heterogeneous persistence storages (purely technical issue).

C: Es war sehr viel Integrationsarbeit nötig bei der Einbindung von Drittprodukten.

? Was ist der Nutzen der erstellten Software?

A: Ziel war E-Collaboration;

B: Users have an integrated environment to work with in the Internet in the context of their business.

? Wurde der Nutzen quantifiziert? Wenn ja, in welcher Form? Wenn nein, warum nicht?

A: Einzelne Kunden machten es und fanden der Return of Investment sei gegeben. Aber es wurde keine Quantifizierung im grossen Stil gemacht.

B: No, benefits were not quantified, at least from the point of view of developers. As far as I know, the company had some measurable targets such as number of users to be reached by certain time.

C: Es wurde versucht, mit Benutzerbefragungen bei etwa zehn Personen eine Auswertung über den Nutzen zu machen, aber es war keine effektive Quantifizierung

Wie wurde das Projekt abgewickelt?

? Nach welcher Methode wurde vorgegangen (z.B. Extreme Programming)?

A: Am ersten evolutionäres Prototyping. Eigene Methode hat sich nach und nach entwickelt.

B: There was no "out-of-box" methodology used. We tried to develop one on its own. We had our own coding guidelines, some common approach for the application development, testing and deployment. Methodology also was evolving in time. At the beginning, it looked more like XP (if there was any methodology used at all at this time). After one year of operation it moved in the direction of RUP.

C: Guerilla Taktik. Eigentlich keine Methode, am ersten eine Anlehnung an RUP Prozesse, aber im Prinzip verlief die Entwicklung chaotisch.

? Wie wurden die Projektmitarbeiter auf die verwendete Methode vorbereitet?

A: Durch interne Schulung. Zudem wurde ein Teilprojekt mit einem Methodikspezialist sauber durchgezogen. Dieses Teilprojekt war dann das Musterbeispiel für die weiteren Projekte.

B: As I said before, people had to adapt to the methodology on the fly. This was not a very big problem for most of the team, since they already had experience with one methodology or another.

C: Gar nicht oder ad hoc.

? Wurden spezielle Werkzeuge verwendet?

A: Kein spezielles Tool. UML Diagramme mit Visio. Datenmodelle direkt mit SQL Statements

B: We used CVS for teamwork. ILogix Raphsody Modeler and Visio as case tools. That are the only things related to methodology I can recall now.

C: Keine, nur Visio für Diagramme.

Grundsatz 1: Einbezug Benutzer

? Wurde dieser Grundsatz eingehalten? *1: B 5: A 6: C*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

B: At the beginning of the project there was no customer at all. People, who were creating specification, were responsible for defining what User needs. During pilot we also had very limited feedback from the customers.

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Durch wiederholte Benutzerinterviews, jeweils nach Aufschaltung eines neuen Release.

C: Einbezug fand teilweise statt, aber vom Projekt her war es das Ziel, auch neue Bedürfnisse überhaupt zu wecken.

? Wer hat die Spezifikation geschrieben?

A: Business Development Team, spezielle Personen (Analytiker), welche diese mit den Marketing Leuten erarbeitet haben. Diese Analytiker waren der IT Abteilung zugeteilt und waren die Schnittstelle zu den Programmierern.

B: Projektleiter.

C: Leiter Entwicklung und Leute, die die Rolle des Anwenders übernommen haben.

? In welcher Form wurde die Spezifikation erstellt?

A: Musterbeispiel diente als Template, dann mit Grafiken (Visio) und bei kritischen Teilen auch Sequenzdiagramme. Für GUI wurde immer mit HTML Beispielen gearbeitet.

B: This changed during the project lifecycle. We started with almost no specifications, when desired featured were described during discussion and presentations, or even were proposed by developers. Later, specifications were prepared in the form of Use Cases.

C: Sehr unterschiedliche Formen, stark abhängig von den Personen, die es gemacht haben. Varianten waren sehr schulbuchmässige, vorbildliche Use Cases, dann HTML Beispiele und Power Point Folien.

? Wurde die Spezifikation im Verlaufe des Projektes angepasst und wenn ja, warum?

B: Specifications were sometimes adapted based on requests from marketing. We also had an impact on specifications, when proposed functionality would be too costly to implement or too computationally intensive.

C: Teilweise wurden sie angepasst, vor allem die, welche formal sauber geschrieben waren. Skizzenhafte Spezifikationen (z.B. Power Point) wurden nicht aktualisiert. Die Zeit dazu war nicht vorhanden.

? Wann erhielten die Anwender erste Programmversionen?

B: etwa nach 30% der Zeit.

? Wie häufig erhielten die Anwender neue Versionen?

A: Neuer Release im Dreimonatszyklus. Alle Monate ein Zwischenrelease.

B: New releases followed old ones with a period of 0.5 to 2.5 month.

? Wie wurde überprüft, ob die Spezifikation eingehalten wurde?

B: Every application had a project leader, responsible for formal acceptance (during the later stages of the project).

C: Es war insofern schwierig, als das gewisse Spezifikationen nur skizzenhaft waren. Wir arbeiteten mit dem Prinzip des Owner einer Idee und dieser hatte dann jeweils die Aufgabe zu überprüfen, ob seine Idee richtig umgesetzt wurde.

Ich finde den Grundsatz "Einbezug Benutzer"
wichtig für das Gelingen eines Projektes: *4: A, 7: B, C*

? Können Sie Ihre Gewichtung begründen ?

B: Software projects are being developed for users, and their satisfaction is the measure of how successful a project was. So, it must be insured that users' requirements and expectations are met. Hence, close cooperation with the user is required, especially when requirements may change dynamically.

C: Unerlässlich, damit man keine Systeme baut, die am Benutzer vorbeigehen.

? Haben Sie weitere Bemerkungen zum Grundsatz "Einbezug Benutzer"?

A: Es ist wichtig. Allerdings ist es wie hier manchmal schwierig, wenn neue Bedürfnisse geweckt werden sollen, dass der Benutzer gar keine Vorstellung hat, was beim Einsatz von neuen Technologien möglich wäre.

Grundsatz 2: Einfach und klein

? Wurde dieser Grundsatz eingehalten? *5: B 6: A 7: C*

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Wir versuchten immer die Aufgaben in einzelne Teilprojekte aufzuteilen. Wichtig ist dass auch Leute bestimmt werden, die den Gesamtüberblick behalten, quasi als einzelnes Teilprojekt.

B: People were divided into small teams of 1-3 developer working on particular subprojects. Code revisions were held to avoid duplication of functionally.

C: Das Projekt war gegliedert in einzelne Applikationen, welche von ein bis zwei Personen entwickelt wurden.

? Enthält das Programm Funktionen, die nie gebraucht werden?

A: Ja schon, allerdings vor allem weil die Benutzerführung schlecht gelöst war und die Funktionen vom Anwender nicht verstanden wurden.

B: Some of the classes became obsolete, after code re-structurisation.

C: Garantiert.

? Falls unötige Funktionalitäten vorhanden sind, warum?

B: Old functionalities were marked as deprecated, and were continuously removed from the code during re-structurizations. They were preserved for compatibility purposes, since it's quite difficult to remove deprecated invocations everywhere, if you have several hundred thousand lines of code.

C: Es hat mindestens eine Codeleiche, niemand hatte Zeit, diese zu entfernen.

? Wieviel Schulung ist nötig, um mit dem Produkt zu arbeiten?

A: Wir dachten es sollte ohne Schulung bedienbar sein, aber dort wären noch Verbesserungen möglich gewesen. Ein gutes Help und eine Quicktour hätte die Schulung unnötig machen sollen, aber das Help wäre noch verbesserungsfähig gewesen (mehr kontextsensitiv).

B: When we had new people joining the team, it was normal to allocate one week for education. After that period, they felt comfortable with the environment and common libraries, and could work on a given application. After another one or two weeks they were aware of the whole system architecture and common techniques used.

C: Das ist abhängig von der Erfahrung eines Benutzers. Jemand der versiert ist im Umgang mit Web Tools, kann sämtliche Funktionen in einem halben Tag erlernen.

? Wie wurde die Arbeit unter den Projektmitarbeitern aufgeteilt?

A: Top down. Chefentwickler gab die Arbeit an vier Teilprojektentwickler weiter, welcher es an ihre Leute weitergaben. Zudem gab es einen Releasemanager.

B: As I said before, people were assigned to small sub-projects, according to their abilities and preferences. A few most experienced developers were able to tailor the system architecture. Subproject can be classified as a set of new related feature to be added into the next release; or as another activity that has to be handled as a separate traceable task.

C: Die Aufteilung erfolgte nach Wissen und Können (Webfactory, DB-Spezialist, etc.). Leute die schon spezialisiert waren in einem Gebiet haben dann dort entsprechende Aufgaben übernommen.

? Führte die gewählte Aufteilung zu Problemen? Wenn ja, welche?

A: Ja. Es gab gewisse Key Personen aus dem Ausland, welche zwischendurch mehrere Wochen nicht verfügbar waren. Das Wissen war auf zuwenig Personen verteilt.

B: When there was bad communication between teams, some efforts were duplicated. As application evolved, changes where required in the common parts, which interfered with other applications. Another problem is that almost all the developers had to deal with user interface and in this area it was very difficult to achieve uniformity.

C: Insofern gab es Probleme, dass wenn eine Person, die für ein bestimmtes Teilstück verantwortlich war, nicht verfügbar war, war es sehr schwierig dort etwas zu korrigieren. Das kam auch daher, dass die Dokumentation relativ schlecht war.

? Wieviele Projektmitglieder kennen den gesamten Sourcecode?

A: Vermutlich etwa drei bis vier Personen waren in der Lage, ein mögliches Problem sofort zu lokalisieren.

B: About 4 people.

C: Ich denke niemand. Es gibt zwei Personen, die einen guten Überblick über alle Module hatten.

Ich finde den Grundsatz "Einfach und klein"
wichtig für das Gelingen eines Projektes: *4: C 6: B 7: A*

? Können Sie Ihre Gewichtung begründen ?

A: In jedem Projekt gibt es soviele Unsicherheitsfaktoren. Zudem ist die Zeit sowieso immer knapp. Wenn es nicht gelingt, die Arbeit in kleine Teile herunterzubrechen gibt es vom Projektcontrolling her schnell einen riesigen Overhead. Zudem werden Probleme schneller erkannt.

B: It's important to have small and well-structured code, so that maintenance of the software becomes easier. New members will be easier to integrate into the team. When code is small and interfaces are well defined, system is less error prone. On the other hand, efficiency should also be always taken into consideration.

C: Nicht alles kann klein gemacht werden. Gewisse Probleme verlangen komplexe Lösungen. So einfach wie möglich aber nicht einfacher sollte sicher immer eingehalten werden.

Grundsatz 3: Qualität in allen Teilen

? Wurde dieser Grundsatz eingehalten? *3: A 4: C 5: B*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Das Marketing fand häufig, die IT Abteilung arbeite unnötig perfekt, aber ich fand die Qualität ungenügend. Um den Zeitplan einzuhalten kam das Testen häufig zu kurz.

B: Most of the important classes had associated test units, so they were well-tested. Testing was also performed on the application and system level. All bugs were reported into the bug tracking system and fixed continuously. Periodical code reviews and re-structurization helped to keep it maintainable.

C: Gewisse Teile wurden sehr sauber realisiert. Andere Teile sind ohne grosse Planung einfach gewachsen.

? Gibt es im Endprodukt bekannte, nicht behobene Fehler?

A: Es gibt Fehler. Diese sind auf einer Bugliste nachgeführt.

B: Yes, the last release had some known issues that were scheduled to be fixed, but the project was stopped.

C: Ja.

? Warum sind die Fehler vorhanden?

A: Wo gearbeitet wird gibt es Fehler.

B: Their impact on the system was not very severe, and they were not easy to fix.

? Warum werden die Fehler nicht behoben?

A: Wir teilten die Resourcen auf in 15-20% für Fehlerkorrektur und der Rest für neue Features. Allerdings wurde die Resourcen für Fehlerkorrektur häufig gebraucht, um auf Releasetermine hin die neuen Features fertigzustellen, und so blieben die Fehler liegen.

C: Das Projekt wurde abgebrochen.

? Wurden regelmässig Tests durchgeführt?

A: Zwei Personen waren nur für Tests zuständig. Sie wurden im Testen auch ausgebildet. Einziges Problem war, dass sie nur Symptome erkennen konnten und die genaue Fehlerlokalisierung wieder Zeit von den Entwicklern in Anspruch nahm.

B: Yes.

C:Ja.

? In welcher Form wurde getestet?

A: Mit Testsoftware, welche über Skripts standardisierte Abläufe simulieren konnte.

B: See one of the previous questions.

C: Mit Hilfe eines Testtools, das automatisierte Tests erlaubte. Zudem gab es Drehbücher für manuelle Tests, wobei diese Drehbücher nicht formal festgelegt waren. Neben dem Zeitdruck gab es ein Sprachproblem mit den zwei für Tests vorgesehenen Personen.

? Gibt es Teile in der Software die neu geschrieben werden sollten?

A: Datenmodell sollte überarbeitet werden, da es gewachsen ist und nicht mehr ideal ist.

B: I would re-write two parts.

C: Ja.

? Warum sollten welche Teile neu geschrieben werden?

B: The first one was an interface to a third-party billing system. At the beginning of the project, nobody knew the bottlenecks of this system, even external consultants. As the consequence, processing of transactions that involved this system was very slow. Solution would be to change the data stored in this system and the method we worked with it.

The second one (user access rights) changed dramatically during the project life cycle. With every new release we had new requirements for this part, and at the end it became quit messy.

C: Das ganze ist gewachsen. Die Entwickler waren verfügbar bevor die Spezifikationen geschrieben waren und so hatte man die Leute irgendwie beschäftigt. So wurde aus Versuchen schlussendlich die definitiven Applikationen mit den entsprechenden Problemen.

? Gab es eine institutionalisierte Form für Design- und Codereview?

A: Codereviews wurden durch Chefentwickler durchgeführt, allerdings in freier Form.

B: We had certain guidelines, and every review was matching code and design against these guidelines.

C: Es gab Codereviews. Wir hatten aber nicht genügend Ressourcen und Zeit, um gute und vollständige Reviews durchzuführen.

? In welcher Form wurden die Reviews durchgeführt?

B: Reviews were performed by analyzing source codes and documentation.

? Ist die Software hinreichend dokumentiert?

A: Ja. Könnte noch verbessert werden.

B: According to our coding guidelines, every class and every public method of the class should be documented. Documenting guidelines were followed in most cases. (Except for the pre-release rush and when some hot-patches were developed).

C: Nein. Die Dokumentation ist sehr unterschiedlich, je nach Applikation.

? Welche Art von Dokumentationen wurden erstellt?

A: Datenmodell, Use Cases, JavaDoc.

B: For the source code we used Javadocs. Applications also had documentation that clarified their general architecture and has class diagrams, use cases, etc. Separate documentation packages were developed for DB and LDAP. System assembling, deployment and trouble-shooting were described in "Portal Deployment Guide".

C: Use Cases, gut dokumentierter Code.

Ich finde den Grundsatz "Qualität in allen Teilen"
wichtig für das Gelingen eines Projektes: *5: A, 7: B, C*

? Können Sie Ihre Gewichtung begründen ?

A: Ist wichtig aber man kann auch in Schönheit sterben.

B: It always worth doing things well from the very beginning. Once something has to be re-implemented, costs increase, and problems demolishes users' trust in the system, making it less attractive.

? Haben Sie weitere Bemerkungen zum Grundsatz "Qualität in allen Teilen"?

A: Wichtiges Qualitätsmerkmal ist auch dass am Schluss das herauskommt, was sich der Benutzer vorstellt.

C: Grundsätzlich ist das wichtig, aber es gibt immer Teile, wo die Qualität nicht so wichtig ist. Periphere Teile oder Wegwerfprogramme wie Interfaces zu anderen Programmen, die ab dem Tag X nicht mehr existieren.

Grundsatz 4: Zeitplan und Ziele

? Wurde dieser Grundsatz eingehalten? *4: B 5: A 6: C*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Die Mitarbeiter haben erst mit der Zeit nach einem Zeitplan gearbeitet, vermutlich vor allem weil ihnen die Erfahrung fehlte.

B: It's difficult to follow any schedule, when scope changes. Besides, situation on the market was changing, marketing requirements were changing, and there was no "von Start bis Ende" plan for this project at all. Though we followed short-time plans for a release (for a few weeks).

? Wie wurde der Zeitplan festgelegt?

A: Schriftlich mit Excel (Grobschätzung). Feinplanung mit MS Project.

B: Schriftlich

C: Es gab eine Featureplanung für die einzelnen Releases. Alle drei Monate gab es ein Release und basierend auf Schätzungen wurde dann rückwärts gerechnet.

? Wie detailliert war der Zeitplan aufgeteilt?

A: Aufgegliedert waren nach Personen.

B: Depending on the activity, from one hour (for deployments) to one week. Usual period for tracking activities was within 1 to 3 days.

C: Detailierungsgrad war wochenweise.

? Waren jedem Mitarbeiter der Zeitplan / Ziele bekannt?

A: Auf Stufe Teilprojektleiter war er sicher bekannt.

B: Yes. For their part of the schedule, at least.

C: Ja.

? Wie wurde die Einhaltung Ziele / Zeitplan überprüft?

B: Once functionality has been tested and accepted by the team of tester, it was considered implemented and corresponding activity closed.

C: Mit jedem Team gab es ein wöchentliches Status Meeting, wo der Stand überprüft wurde.

? Wie wurden auf Abweichungen Realität <-> Zeitplan reagiert?

A: Funktionalitäten wurden weggelassen. Zudem wurde die Zeit für die Tests verkürzt, dass heisst Freeze Termine nicht eingehalten.

B: The reasons for these deviations were discovered. If the activity was on critical path, extra resources were assigned. The estimates for the future were updated.

C: Es gab zu unklar definierte oder zu aufwendige Features, welche dann während einer Phase zurückgestellt werden mussten.

Ich finde den Grundsatz "Zeitplan"
wichtig für das Gelingen eines Projektes: *6: A, B 7: C*

? Können Sie Ihre Gewichtung begründen ?

A: Es ist wichtig und Bestandteil der drei Komponenten Qualität - Zeit - Kosten.

B: Yes, it's very important to be on track. But I would change slightly your Time-principle to take into account two things: 1. It is not always necessary to track activities on a daily basis. Granularity should be chosen based on the programmer's qualification and type of activity. 2. It is not always possible to have „Start to End" plan for big projects, because environment may change and most like this plan will be inaccurate. This should be the case, however for short phases of such projects (an order of month long).

? Haben Sie weitere Bemerkungen zum Grundsatz "Zeitplan"?

A: Zeitplan ist etwas vom Schwierigsten, es ist nur mit Erfahrung lernbar.

Grundsatz 5: Spass der Beteiligten

? Wurde dieser Grundsatz eingehalten? 5: C *6: B*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

C: Zwischendurch war der Zeitdruck einfach zu gross.

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

B: Team members were enthusiastic about what they are doing. Project was a challenge. Climate was friendly and we felt like a team.

C: Das Arbeitsklima war sehr gut.

? Was war Ihre wichtigste Motivation bei der Arbeit an diesem Projekt? Warum?

A: Neuste Technologie, sehr interessante Idee, die etwas bewegen würde in der Internet Landschaft.

B: As I already said, this project was a very big one. It was a challenge for every programmer.

C: Etwas von dieser Grösse, in so kurzer Zeit und einem so dynamischen Umfeld zu realisieren.

? Gab es für sie persönlich wichtige Ziele, die nicht erreicht wurden? Welche? Warum?

A: Weniger Eigenentwicklungen, mehr der Einsatz von Komponenten, die sich dann auch selber weiterentwickeln. Vielleicht ist das bei Internet Services auch nicht möglich.

B: I considered the success of the Project as one of my personal goals, since one of the biggest awards for a programmer is when his product finds its users and makes them happy.

C: Das Projekt gibt es nicht mehr. Es gab zuwenig Raum für Ideen in Bezug auf die Kreativität bei der Software Architektur.

? Was hat Ihnen bei diesem Projekt am wenigsten gefallen? Warum?

A: Hektik, welche bei der Qualität viele Abstriche verlangte. Zudem schafften wir es nicht, genügende Benutzerfreundlichkeit hinzubringen. Vermutlich fehlten die Leute, die für diese Arbeit geeignet wären.

B: Frequent changes of marketing policy (with all the corresponding consequences on development, such as wasted efforts and rush before new releases).

C: Der enorme Zeitdruck war ziemlich belastend.

? Was hat bei diesem Projekt besonders gut funktioniert? Warum?

A: Gutes Team, Termin und Kosten wurden eingehalten.

B: The climate within development department was very nice.

C: Das Entwicklerteam war gut eingespielt, alle arbeiteten in die gleich Richtung.

? Wenn Sie rückblickend Ihre Rolle im Projekt ändern könnten, was würden Sie ändern?

A: nichts.

B: I would try to improve development methodology from the very beginning.

C: Meine Rolle war in den gesteckten Rahmenbedingungen das Optimum. Ich würde das Projekt ändern.

? Gab es im Laufe des Projektes Konflikte unter den beteiligten Personen?

A: Es war ein ständiger Kampf zwischen Marketing und IT. Die Arbeitsweise des Marketings war häufig oberflächlich und zu stark auf Kundenbedürfnisse ausgerichtet.

Auch zwischen den Mitarbeitern in der IT Abteilung traten kleinere Konflikte auf, wie das wohl überall der Fall ist.

B: Yes, there were some problems as a result of bad communication or personal ambitions.

C: Ja, es gab Animositäten zwischen einzelnen Personen, die Mühe hatten, gute Ideen von anderen Leuten zu akzeptieren oder sogar einzugestehen, dass andere Leute bessere Ideen haben.

? Wie wurden aufgetretene Konflikte gelöst?

A: Leute, die sich gegenseitig nicht vertrugen wurden in verschiedene Entwicklerteams zugeteilt. Das klappte recht gut, da es viele einzelne Teilprojekte gab. So konnte viele Spannungen abgebaut werden.

Zwischen Marketing und IT wurde versucht die Prozesse mehr zu formalisieren.

B: By assigning potentially conflicting people to non-related activities, so that their competition would not have an impact on the job quality. Team building events and other common activities were also useful to set up a friendly environment.

C: In einem Fall wurde die betroffene Person temporär aus dem Team entfernt. Er kam später wieder dazu und arbeitete an anderen Teilprojekten.

? Gab es personelle Wechsel im Projektteam? Gründe ?

A: Nicht viele. Es wurde einfach versucht Leute mit verschiedener Arbeitsweise in unterschiedliche Teams einzuteilen.

B: Yes, and its normal since project went for more than 17 month. One person was changed because it was virtually impossible to set a communication with him, and he did not integrate into the team. Some people left the team because they applied for another job. I think that some more people, who worked on the routine parts of the system and became bored, would leave if the project had to go on.

C: Es gab Wechsel aus verschiedenen Gründen (Heimweh, andere suchten noch bessere Stellen in der USA, familiäre Gründe).

? Gab es gesellschaftliche Anlässe innerhalb des Projektteams? Welche? Von wem initiert?

A: Ja, Kegeln, Grillparties, Bierrunde nach Releases, initiert durch Projektleiter und Mitarbeiter.

B: I'm not quite sure if I got the question right. If „gesellschaftliche Anlässe" = „team building events", then yes. We had several events during the project. Official ones were initiated by the company administration and usually took place after big releases.

C: Ja, meiner Ansicht nach zuwenig. Wir haben zuviel gearbeitet und solche Anlässe kamen zu kurz. Initiert wurden die Anlässe typischerweise vom Mangement.

? Nahmen Sie an solchen Anlässen teil? Warum? Warum nicht?

A: Ja, es ist wichtig, dass man sich versteht.

B: Yes.

C: Sicher.

Ich finde den Grundsatz "Spass der Beteiligten"
wichtig für das Gelingen eines Projektes: *5: A, 7: B, C*

? Können Sie Ihre Gewichtung begründen ?

B: Programming is a creative activity and requires devotion from team members. It cannot be easily controlled, like some routine jobs. Unless people are motivated, results will not be good.

C: Die Arbeit muss Freude machen. Es braucht Raum für Kreativität. Es bringt aber nichts immer die wunderbare tolle Lösung finden zu wollen, den das dauert einfach zu lange.

? Haben Sie weitere Bemerkungen zum Grundsatz " Spass der Beteiligten "?

A: Freude und gutes Arbeitsklima ist wichtig. Spass würde ich nicht zu hoch gewichten. Das Wort ist auch negativ belastet. Freude finde ich eine treffenderen Ausdruck.

Fragebogen / Interview Projekt 4

Allgemeine Angaben zum Projekt

Ihre Rolle im Projekt: *A, C: Entwickler, B: Projektleiter*

Fachgebiet: *Internet/Intranet Applikation*

Zeitdauer des Projektes: Beginn am: *..... Jan 1999* Ende am: *..... Dez 2001*

Anzahl beteiligte Personen: Total: *.....13* davon Entwickler: *.....7-9/5*

Aufwand in Arbeitstagen: Total: ..... davon Entwicklung: .....

Umfang des Projektes: Anzahl Applikationen: *1 (aufgeteilt in ca. 25 Funktionen)*

Anzahl Zeilen Sourcecode: Seite Mainframe (Cobol): mehrere 100'000 Zeilen Code

Anzahl Klassen: dieses Projekt: *..690* eigene Library: *..100* Drittlibraries: *..... 10*

Programmiersprache(n): *Java, Java-Script, Cobol*

Entwicklungsumgebung(en): *Visual Age for Java*

War das Projekt erfolgreich?

Termine wurden eingehalten: *4: A 5: C, 6: B*

Kostenrahmen wurde eingehalten: *5: A, C 6: B, C*

Software ist fehlerfrei: *3: A 6: B, C*

Software erfüllt die Benutzeranforderungen: *6: B, C 7: A*

Software ist wartbar: *6: A, B, C*

? Welche Rahmenbedingungen machten das Projekt speziell einfach?

A: Keine

B: Keine. Ausser vielleicht dass es ein gutes Team war und die Komplexität der Aufgabenstellung überschaubar war.

? Welche Rahmenbedingungen machten das Projekt speziell kompliziert?

A: Daten enthalten nicht nur aktuellen Stand sondern sind historisiert vorhanden. Einsatz neuer, für uns unbekannter Technologie (Java Server Pages anstelle C++).

B: Es war ein Pilotprojekt, erste Internet/Internetanwendung für die beteiligten Projektmitarbeiter.

C: Die ständig wachsenden Benutzeranforderungen und Spezialwünsche war sicher erschwerend.

? Was ist der Nutzen der erstellten Software?

A: Internetpräsenz / Auskunftssystem.

? Wurde der Nutzen quantifiziert? Wenn ja, in welcher Form? Wenn nein, warum nicht?

A: Weiss nicht. Vermutlich fand das im Rahmen des Konzeptes statt. Eine Quantifizierung ist schwierig, da man genauere Informationen haben müsste, wie das Produkt eingesetzt wird.

B: Die Quantifizierung ist Aufgabe des Kunden. Eine genaue Quantifizierung ist im aktuellen Fall schwierig, da es sich um etwas neues handelt. Man auch weiss, dass noch nicht alle Leute, dieses neue System nutzen sondern weiterhin konventionell arbeiten.

Wie wurde das Projekt abgewickelt?

? Nach welcher Methode wurde vorgegangen (z.B. Extreme Programming)?

A: Nach Hermes.

B: Nach Hermes.

C: Nach Hermes.

? Wie wurden die Projektmitarbeiter auf die verwendete Methode vorbereitet?

A: Ist bei allen Mitarbeitern bekannt. Es werden auch interne Ausbildungen angeboten.

? Wie wurde geprüft, ob die Methode richtig angewendet wurde?

A: QS Stelle, welche Reviews durchführt.

B: QS Richtlinien gewährleisten die korrekte Verwendung der Methode.

? Wurden spezielle Werkzeuge verwendet?

A: Kein spezielles Tool. Es werden Word Vorlagen und Flow Charts verwendet.

B: Keine speziellen Tools ausser Frontpage und Visual Age for Java.

Grundsatz 1: Einbezug Benutzer

? Wurde dieser Grundsatz eingehalten? *7: A, B, C*

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Nicht nur ein Benutzer, sondern ein Benutzerteam arbeitet eng mit den Entwicklern zusammen. Das Projekt ist in Teilprojekte aufgeteilt. Jedes Teilprojekt hat ein eigenes Benutzerteam, wo der Leiter dieses Teams zwar schon der Hauptansprechpartner ist, aber er kann nicht alleine entscheiden.

B: Schon in der Konzeptphase war ein Benutzerteam dabei, welches die betrieblichen Bedürfnisse formuliert hat. Mit Prototypen haben wir Beispiele durchgespielt, um auch mögliche Probleme zu klären, die sich nicht so einfach verbal beschreiben lassen.

C: Einbezug Benutzer war in diesem Projekt ein zentrales Anliegen von Anfang an.

? Wer hat die Spezifikation geschrieben?

A: Teilprojektleiter schrieb Spezifikation der Teilprojekte.

B: Projektleiter.

C: Projektleiter und Anwender.

? In welcher Form wurde die Spezifikation erstellt?

A: Word Dokumente, Print Screens, HTML und Java/Swing Prototypen.

B: Gemäss Hermes und nach internem Vorgehensmodell ist die Struktur der Dokumente vorgegeben.

C: Word Dokumente.

? Wurde die Spezifikation im Verlaufe des Projektes angepasst und wenn ja, warum?

A: Ja, aufgrund zusätzlicher Anforderungen aus dem Benutzerteam.

B: Es gab nur kleinere Anpassungen. Auch nach der definitiven Einführungen waren sehr wenig Korrekturen und Anpassungen nötig.

C: Es gab laufend Anpassungen, aus Gründen von der verwendeten Technologie (Machbarkeit) aber auch wegen geänderten Benutzerwünschen.

? Wann erhielten die Anwender erste Programmversionen?

A: Die Endanwender wurden spät einbezogen, ausser natürlich das Benutzerteam. Produktiv zum Einsatz kam die Software erst, als sie fertig war. Nur einzelne Teile (französische Übersetzung) wurden erst nachträglich ausgeliefert.

B: Das ganze wurde als Einheit nach der Fertigstellung eingeführt. Die interne Entwicklung ging schrittweise, das heisst es wurden die einzelnen Teilprojekte entwickelt und mit dem Benutzerteam auf Brauchbarkeit überprüft.

? Wie häufig erhielten die Anwender neue Versionen?

A: Wenig. Mit Intranet Applikationen ist es einfacher, da diese nicht überall installiert werden müssen.

? Wie wurde überprüft, ob die Spezifikation eingehalten wurde?

A: Mit Abnahmeprotokoll mit Gegenüberstellung der einzelnen Punkte. Das Abnahmeprotokoll wurde von den Teilprojektleitern vorbereitet und der Kunde nimmt es ab und kann es mit eigenen Bemerkungen ergänzen.

B: Mit Hilfe des Funktionenrasters aus der Spezifikation. Der Entwickler erstellt eine einzelne Funktion, diese wird vom Projektleiter getestet und anschliessend noch durch das Benutzerteam überprüft.

Ich finde den Grundsatz "Einbezug Benutzer"
wichtig für das Gelingen eines Projektes: *7: A, B, C*

? Können Sie Ihre Gewichtung begründen ?

A: Es ist äusserst wichtig. Man spricht einfach nicht die gleiche Sprache, gleiche Worte haben eine verschiedene Bedeutung für Entwickler und Anwender. Bei der direkten Zusammenarbeit im Projekt, ist es einfacher, solche Differenzen zu erkennen.

B: Ich denke, beim Bau von Individualsoftware (wie in diesem Fall), ist es sehr wichtig, dass der Benutzer gut einbezogen wird. Der Benutzer sagt was er will und wir setzen es um. Bei Standardsoftware ist das unter Umständen anders, da wird der Markt entscheiden.

C: Dieser Grundsatz hat aus meiner Erfahrung gesehen höchste Priorität.

? Haben Sie weitere Bemerkungen zum Grundsatz "Einbezug Benutzer"?

A: Die Zusammenarbeit mit einem einzelnen "Key-User" birgt die Gefahr, dass eine Software am Schluss genau auf die Bedürfnisse dieses Anwenders zugeschnitten ist und für eine grössere Benutzergruppe nicht mehr eine optimale Lösung darstellt, zum Beispiel weil sie zu kompliziert ist.

Ob auf Wünsche eingegangen werden kann, ist immer eine Frage des Zeitpunktes. Gerade wenn bei einem Projekt der finanzielle und zeitliche Rahmen einmal festgelegt wurde, sind grössere Änderungen äusserst schwierig. Dies auch, weil das Benutzerteam erst im Laufe der Zusammenarbeit genau herausfindet, was eigentlich nötig ist. Wenn man dann nicht mehr auf Änderungswünsche eingehen kann, erstellt man eine Applikation, die schlussendlich niemand braucht.

B: Der Einbezug der Benutzer bei einem Projekt führt dazu, dass diese sich mit dem Projekt identifizieren und auch Verantwortung dafür übernehmen.

C: Wünsche, die wachsen. Je mehr der Benutzer sieht erhält in Form von Bildschirmmasken und Prototypen, desto mehr Begehrlichkeiten entstehen.

Grundsatz 2: Einfach und klein

? Wurde dieser Grundsatz eingehalten? *4: A 6: B, C*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Ist sicher ein Ziel, aber nicht immer realisierbar.

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Wir machten eine Aufteilung in Teilprojekte, welche jeweils eine Applikation umfassten. Die Teilprojektteams umfassten drei bis fünf Leute.

B: Aufteilung erfolgte nach dem Funktionsraster. Es sind abgeschlossene, nicht sehr grosse Einheiten.

C: Wir versuchen immer Projektteams so klein wie möglich zu halten.

? Enthält das Programm Funktionen, die nie gebraucht werden?

A: Keine

B: Keine, vielleicht solche, die wenig verwendet werden, aber dies mehr, weil eine grundsätzliche Umgewöhnung der Benutzer nötig ist.

C: Nein, es hat nur verwendete Funktionen.

? Wieviel Schulung ist nötig, um mit dem Produkt zu arbeiten?

A: Das Produkt ist so ausgelegt, dass keine Schulung nötig sein sollte.

B: Wenig, es wurden nur Informationsveranstaltungen mit 100-150 Leuten durchgeführt. Die Praxis zeigt, dass das Ziel, ein einfach und intuitiv bedienbares System zu erstellen, erfüllt wurde. Allerdings würden Systemteile, wo eine grundsätzliche Ablaufänderung beim Benutzer nötig wäre, besser genutzt, wenn eine vertiefte Schulung gemacht worden wäre. Aus Kosten- und Zeitgründen war das aber nicht möglich.

C: Die nötige Schulung erachte ich als minim.

? Wie wurde die Arbeit unter den Projektmitarbeitern aufgeteilt?

A: Aufteilung erfolgte in Abhängigkeit der Technologie (Cobol auf Hostseite, Java auf Clientseite).

B: Es gab eine klare Aufteilung. Jeder Entwickler erhielt genau spezifizierte Funktionen und entwickelte diese selbstständig. Die Integration erfolgt nach Überprüfung durch den Projektleiter.

C: Aufgeteilt nach Kenntnissen der Entwickler.

? Was waren die Gründe für die gewählte Aufteilung?

A: Kenntnisse der Technologien.

? Führte die gewählte Aufteilung zu Problemen? Wenn ja, welche?

A: Probleme gab es in den Details bei den Schnittstellen. Zusätzlich erschwert wurde die Arbeit, weil der Teil in Cobol durch eine Drittfirma realisiert wurde. Flexibles Reagieren ist damit schwieriger.

B: Es tauchten keine Probleme auf.

C: Probleme gab es wegen der Aufteilung in klassische Host Entwickler und neue Technologie. Noch immer besteht zwischen diesen zwei Gruppen ein Kommunikationsproblem, da sie nicht die gleiche Sprache sprechen. Gerade bei Schnittstellen, besteht so immer die Gefahr, dass man nebeneinander vorbei entwickelt. Viel geschieht hier über "learning by doing".

? Wieviele Projektmitglieder kennen den gesamten Sourcecode?

A: Keiner, da auch eine Drittfirma involviert war. Überblicke sind nur auf Konzeptebene vorhanden.

B: Niemand. Es wäre in diesem Projekt fast nicht möglich, den mit Java und Cobol Programmen kommen zwei verschiedene Welten zusammen. Es gibt etwa je eine Person die den Java oder den Cobol Teil überblickt.

Ich finde den Grundsatz "Einfach und klein"
wichtig für das Gelingen eines Projektes: *6: A, B, C*

? Können Sie Ihre Gewichtung begründen ?

B: Eine Applikation ist nur dann überschaubar, wartbar und auf andere Leute übertragbar, wenn sie klar und einfach aufgebaut ist.

C: Ist anzustreben, aber nicht immer möglich.

? Haben Sie weitere Bemerkungen zum Grundsatz "Einfach und klein"?

A: Je mehr man es aufteilen kann, desto besser kann man es "packen" und man behält den Überblick.

Refactoring wäre sicher gut, wird aber nie gemacht, weil es Zeit und Geld kostet. Im kleinen ja, aber grundsätzlich haben wir es nicht gemacht.

B: Ich würde sagen, möglichst einfach und klein . Applikationen werden manchmal einfach zu kompliziert und monolythisch gebaut. Dies geschieht häufig, wenn einzelne dominante Entwickler in einem Projekt involviert ist. Probleme gibt es dann, wenn diese Person nicht verfügbar ist.

C: Gerade auf Hostseite werden Programme häufig dann nicht einfach und klein, wenn man durch äussere Umstände (z.B. Gesetzesänderungen) Anpassungen vornehmen muss. Schnell entstehen dann schwer wartbare und durchschaubare Moloche, welche meistens auch stark auf einen Entwickler bezogen sind.

Grundsatz 3: Qualität in allen Teilen

? Wurde dieser Grundsatz eingehalten? *5: A 6: C*

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Sichergestellt durch interne Reviews.

B: Wir haben eine erste in sich abgeschlossene Teilfunktion als Musterlösung realisiert. Diese diente dann als Vorlage für viele weiteren Funktionen. Dies war sicher besser, als wenn mehrere Teams gleichzeitig mit einer Funktion begonnen hätten. Dieses Vorgehen hat sich bewährt.

C: Es war sicher ein Ziel, wie genau es herausgekommen ist kann ich nicht beurteilen. Abweichungen von diesem Ziel kamen wegen Zeitdruck und geänderten Benutzeranforderungen, welche nicht mehr in das ursprüngliche Raster passten.

? Gibt es im Endprodukt bekannte, nicht behobene Fehler?

A: Es gibt Fehler.

B: Es gibt keine bekannten Fehler.

C: Die mir bekannten Schnittstellen sind fehlerfrei.

? Warum sind die Fehler vorhanden?

A: Wo gearbeitet wird gibt es Fehler.

? Warum werden die Fehler nicht behoben?

A: Die Fehler betreffen Funktionen, die selten verwendet werden und haben auf das Gesamtsystem keine Auswirkungen.

? Wurden regelmässig Tests durchgeführt?

B: Ja.

C: Es wurden sehr häufig Tests durchgeführt.

? In welcher Form wurde getestet?

A:.Durch das Benutzerteam mit Hilfe von Testprotokollen

B: Einzeltests beim Entwickler, Kettentests beim Projektleiter, dann Tests im Benutzerteam. Der Praxistest war dann bereits im produktiven Einsatz. Die Leute konnten ihre Daten ca. drei Wochen vor dem offiziellen Aufschalttermin im Internet eingeben. So konnten auch diese letzten Live Tests ohne Zeitdruck durchgeführt und noch kleinere Probleme behoben werden. Es gab kein automatisiertes Testverfahren.

C: Stufenweise auf den einzelnen Entwicklungsebenen: lokal, Einzeltest, Kettentest, Funktionstest. Aus technischen Gründen wurde bei diesem Projekt sehr früh in der Produktion getestet, weil dort die Reaktionszeiten ganz anders sind. Das ist ein Phänomen, dass wir in unserer Umgebung sehr häufig feststellen: Es wird früh und direkt in der Produktion getestet, was aus Zeit- und Kostengründen vielfach das vernünftigste ist.

? Gibt es Teile in der Software die neu geschrieben werden sollten?

A: Im Moment nicht, wird aber sicher kommen, sobald die Anforderungen ändern.

B: Ich denke nein.

? Gab es eine institutionalisierte Form für Design- und Codereview?

A: Gibt es in Form einer Qualitätssicherungs Abteilung.

B: Nein.

C: Keine Codereviews. Hostseitig wird über Hermes ein Review des Designs sichergestellt.

? In welcher Form wurden die Reviews durchgeführt?

B: Wir achteten auf Einheitlichkeit. Mit der Musterlösung hatte man Aufbau und Design ziemlich gut festgelegt.

? Warum fanden keine Reviews statt?

C: Codereview wäre sicher sinnvoll, ist aber aus Zeit- und Kostengründen schlicht nicht möglich. Es ist auch schwierig, nach Jahren so etwas einzuführen. Es bleibt dann bei kleinen Formalismen wie ein bisschen Einrücken hier und eine zusätzliche Kommentarzeile da.

? Ist die Software hinreichend dokumentiert?

A: Ja.

B: Ja.

C: Ja.

? Welche Art von Dokumentationen wurden erstellt?

A: Beschreibung der Schnittstellen und JavaDoc für Java Programme. Es bestehen nicht alle Dokumente nach Hermes. Wir haben ein internes Modell, welches auf Hermes basiert und den eigenen Bedürfnissen angepasst ist. Problem der Dokumentation ist, man macht es zu spät und hat dann keine Zeit und Geld mehr.

B: Gegeben durch die Struktur nach Hermes.

Ich finde den Grundsatz "Qualität in allen Teilen"
wichtig für das Gelingen eines Projektes: *6: A, C 7: B*

? Können Sie Ihre Gewichtung begründen ?

A: Ist sicher wichtig, den Fehler werden sich früher oder später immer negativ auswirken.

B: Ich denke, genau hier produziert man sehr viel Unmut, wenn man die Qualität nicht sicherstellen kann. Qualität nachträglich zu erreichen ist immer schwierig und fast nicht machbar.

? Haben Sie weitere Bemerkungen zum Grundsatz "Qualität in allen Teilen"?

A: Der Ansatz Prototypen korrekt zu machen ist durchaus richtig und findet nicht immer statt.

B: Es gibt immer ein Spannungsfeld. Qualität heisst man hat Zeit und Geld etwas zu machen. Zeitdruck und Geldmangel ist in jedem Projekt vorhanden und es ist immer ein abwägen zwischen Kosten und Qualität. Man muss einen Mittelweg finden.

C: Qualität soll nicht das Maximum, sondern das Optimum in dem Rahmen, wo man sich bewegt, sein. Das heisst nicht in jedem Fall eine Null Fehler Lösung sondern es kann auch eine 80% Lösung sein. Aber es muss mit dem Kunden/Benutzer diskutiert und festgelegt werden.

Grundsatz 4: Zeitplan und Ziele

? Wurde dieser Grundsatz eingehalten? *2: A 6: B, C*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Der Ersteller des Zeitplans war sehr theoretisch orientiert. Beim Erstellen des Zeitplans waren weder die Details noch die zu verwendende Technologie bekannt.

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

B: Das Projekt wurde termingerecht eingeführt. Es gab keine Probleme mit den geplanten Meilensteinen.

C: Es ist wichtig, dass Meilensteine gesetzt werden. Der Grundsatz wurde eingehalten, abgesehen von Verschiebungen, die sich durch neue Anforderungen ergeben haben.

? Wie wurde der Zeitplan festgelegt?

A: Schriftlich mit Gesamtprojektplan nach Hermes.

B: Schriftlich.

? Wie detailliert war der Zeitplan aufgeteilt?

A: Grobplan mit monatlicher Aufteilung und im Detailplan pro Woche.

B: Die einzelnen Funktionen wochenweise.

? Waren jedem Mitarbeiter der Zeitplan / Ziele bekannt?

A: Ja,ist bekannt und immer für alle zugänglich.

B: Ja.

C: Immer allen bekannt, eigentlich auf den Tag genau. Zudem hat jeder seine eigene Arbeit danach eingeteilt und auch sofort reagiert, wenn er Probleme erkannte.

? Wie wurde die Einhaltung Ziele / Zeitplan überprüft?

B: Mit Hilfe von Meilensteinen.

? Wie wurden auf Abweichungen Realität <-> Zeitplan reagiert?

A: Die massive Abweichung wurde durch eine Task Force untersucht und man beschloss eine Technologieänderung von Java/Swing zu Java Server Pages.

C: Unterschiedlich. Primär wurde versucht, für zusätzliche und unklare Anforderungen, beim Kunden auch mehr Zeit zu erhalten.

Ich finde den Grundsatz "Zeitplan"
wichtig für das Gelingen eines Projektes: *6: A, B, C*

? Können Sie Ihre Gewichtung begründen ?

A: Ohne Zeitplan ist ein Projekt gar nicht realisierbar.

B: Häufig ist es so, dass am Anfang eines Projektes sehr viel Zeit vorhanden ist und am Schluss dann keine mehr. Ohne Meilensteine ist es gar nicht möglich, die Kontrolle über ein Projekt zu behalten. Der Planungsaufwand soll aber auch in einem Rahmen bleiben, Tagespläne sind sicher nur für spezielle Arbeiten nötig.

C: Ein Zeitplan ist wichtig, muss immer dynamisch gehalten werden.

? Haben Sie weitere Bemerkungen zum Grundsatz "Zeitplan"?

A: Der Ersteller des Zeitplans braucht hohe technologische Kompetenz oder der Ersteller muss auf Leute mit dieser Kompetenz zurückgreifen können.

Nicht einhalten von Zeitplänen heisst immer auch zusätzliche Kosten. Auch unter diesem Aspekt ist ein Zeitplan äusserst wichtig.

B: Zeitplan ist nicht alles. Andere Massnahmen wie die Qualitätsüberprüfung sind ebenso wichtig.

C: Fast wichtiger als, dass ein Zeitplan und Meilensteine definiert werden, ist die laufende Kontrolle und Anpassung dieser Daten. Das darf nicht vernachlässigt werden. Wobei gerade bei absolut fixen Endterminen kann das Probleme mit dem Kunden geben.

Grundsatz 5: Spass der Beteiligten

? Wurde dieser Grundsatz eingehalten? *6: A, B 7: C*

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Ich persönlich finde das Arbeitsklima hier gut.

B: Die Herausforderung, etwas mit neuen Technologien zu machen war sehr motivierend.

C: Das Arbeitsklima, die Räume, die Freiheiten und Kompetenzen stimmen in dieser Firma.

? Was war Ihre wichtigste Motivation bei der Arbeit an diesem Projekt? Warum?

A: Die Herausforderung war der Einsatz der für mich neuen Technologie.

B: Eine brauchbare Internetanwendung anzubieten und zu sehen, dass es mit den Mitarbeitern und dem Benutzerteam gut funktioniert.

C: Meine Motivation ist, dass es auch in unserem kleinen Umfeld, wo jeder jeden kennt möglich ist, mit einem kleinen Baustein einen Beitrag zu leisten, dass ein grosses Projekt zustande kommt.

? Gab es für sie persönlich wichtige Ziele, die nicht erreicht wurden? Welche? Warum?

C: Ein Ziel für mich war, etwas mehr mit neueren Technologien arbeiten zu können. Dazu hat die Zeit in diesem Projekt nicht gereicht.

? Was hat Ihnen bei diesem Projekt am wenigsten gefallen? Warum?

A: Die falschen Schätzungen von Kosten und Terminen wirkt sich unangenehm auf die Arbeit aus, weil man sich dauernd gegenüber Management und Kunde rechtfertigen muss.

B: Kleinere Probleme gibt es immer, bei uns am ersten mit externen Stellen, wo der Kontakt nicht so direkt war wie mit Leuten im Haus. Hier ist es auch schwierig etwas zu ändern.

C: Die fehlende Zeit, neue Technologien kennen zu lernen.

? Was hat bei diesem Projekt besonders gut funktioniert? Warum?

A: Die Zusammenarbeit mit dem Kunde hat sehr gut funktioniert, weil wir diesen auch sehr gut kennen.

C: Die Zusammenarbeit im Team und der Kontakt mit den Benutzern.

? Wenn Sie rückblickend Ihre Rolle im Projekt ändern könnten, was würden Sie ändern?

A: Ich würde das Projekt unter gleichen Voraussetzungen (Kosten / Termine) dieses Projekt ablehnen. Das der Zeitplan nicht eingehalten werden kann war voraussehbar und im Nachhinein hätte man härter darauf bestehen sollen, dass dieser frühzeitig angepasst wird.

B: Nein. Man lernt sicher bei jedem Projekt und hat mehr Erfahrungen, aber ich sehe konkret nichts, was hier gerade geändert werden müsste.

C: Ich hätte gerne mehr Zeit, dass ganze Projekt zu kennen und nicht nur die Schnittstellen zur Hostseite.

? Gab es im Laufe des Projektes Konflikte unter den beteiligten Personen?

A: Ja, Differenz zwischen der Person die das Projekt am Anfang konzeptionell betreut hat und den Leuten, die schlussendlich entwickelten. Der Konflikt entstand vor allem aus den unterschiedlichen Ansichten bezüglich des Zeitplans.

B: Nein. Es war ein relativ kleines Team und alle gut motiviert.

C: Nein, keine mir bekannten.

? Wie wurden aufgetretene Konflikte gelöst?

A: Eigentlich nicht. Der betroffene Mitarbeiter arbeitet heute in einer anderen Abteilung.

? Gab es personelle Wechsel im Projektteam? Gründe ?

A: Ja, durch natürliche Fluktuation. Nicht nur bei uns sondern auch neue Ansprechpartner beim Kunden, was auch zu Problemen führen kann.

B: Nur ein Abgang, gegeben durch äussere Umstände.

C: Natürliche Abgänge. Zugänge, weil zusätzliche Ressourcen für das Projekt nötig waren.

? Gab es gesellschaftliche Anlässe innerhalb des Projektteams? Welche? Von wem initiert?

A: Ja, im Rahmen des Projektes, initiert vom Projektleiter, zum Beispiel gemeinsames Mittagessen.

B: Nein, vielleicht zuwenig, einfach das was im Rahmen unserer Abteilung organisiert wird. Es war in diesem Projekt auch nicht nötig und es gab hier keinen Handlungsbedarf.

C: Ich weiss es nicht. Ich war nie dabei.

? Nahmen Sie an solchen Anlässen teil? Warum? Warum nicht?

A: Ja, die Frage stellt sich nicht, weil die Anlässe eigentlich während der Arbeit stattfanden .

C: Ich nahm nicht teil, aus zeitlichen Gründen und weil ich zuwenig involviert war. Dieses Projekt war für mich eines unter vielen.

Ich finde den Grundsatz "Spass der Beteiligten"
wichtig für das Gelingen eines Projektes: *5: B 6: A 7: C*

? Können Sie Ihre Gewichtung begründen ?

A: Aus jedem Mitarbeiter ist etwas herauszuholen, wenn es ihm Spass macht.

B: Es braucht Freude bei der Arbeit aber das gibt es nicht immer, da in jedem Projekt Zeit- und Kostendruck da ist.

? Haben Sie weitere Bemerkungen zum Grundsatz " Spass der Beteiligten "?

B: Spass ist vielleicht ein schlechtes Wort. Ich würde es eher als Freude und Motivation bezeichnen.

Fragebogen / Interview Projekt 5

Allgemeine Angaben zum Projekt

Ihre Rolle im Projekt:	*A: Projektleiter, B: Entwickler (DB), C: Entwickler (Host), D: Entwickler (Run Streams)*	
Fachgebiet:	*Business Software*	
Zeitdauer des Projektes:	Beginn am:..... Jan. 2001	Ende am: ..... Sept. 2001
Anzahl beteiligte Personen:	Total: *15*	davon Entwickler: *9*
Aufwand in Arbeitstagen:	Total: ..... *700 Tage*	davon Entwicklung:..... *550 Tage*
Umfang des Projektes:	Anzahl Applikationen: *1*	
Anzahl Zeilen Sourcecode:	*mehrere 10'000 Zeilen* *18 Host Programme, ca. 30 PL/SQL Programme*	
Programmiersprache(n):	*Cobol 74/85, PL/SQL, Unix Shell Scripts*	
Entwicklungsumgebung(en):	*CodeWright (für Cobol), SQL Navigator (Quest)*	

War das Projekt erfolgreich?

Termine wurden eingehalten:	*2: A, C, D 3: B*
Kostenrahmen wurde eingehalten:	2: C *4: B 5: A 6: D*
Software ist fehlerfrei:	*6: A, B, C, D*
Software erfüllt die Benutzeranforderungen:	*6: B, C 7: A, D*
Software ist wartbar:	*5: B, D 6: A, C*

? Welche Rahmenbedingungen machten das Projekt speziell einfach?

A: Keine.

B: Die DB Technologie war festgelegt: Serverseitig Oracle und hostseitig Unisys DBMS.

D: Es gab bereits ein Produkt und man kannte deshalb die Benutzeranforderungen bereits sehr gut.

? Welche Rahmenbedingungen machten das Projekt speziell kompliziert?

A: Das Know-How der Mitarbeiter war zu Beginn des Projektes nicht vorhanden, da es nach der Analysephase Abgänge gab.

B: Die Kommunikation zwischen Host Leuten und uns. Die verwendete Terminologie ist sehr unterschiedlich. Das wird beim Bestimmen von Schnittstellen schwierig.

C: Wegen des riesigen Datenvolumens, das jeden morgen transferiert werden muss (bis zu 12GB), traten im Laufe der Arbeit Geschwindigkeitsprobleme auf, deren Lösung einiges an Zeit in Anspruch nahm.

D: Die neuen verwendeten Technologien für den Dateitransfer. Das war auch der Hauptgrund für die zeitliche Verzögerung im Projekt, die neuen Technologie mussten zuerst erlernt werden.

? Was ist der Nutzen der erstellten Software?

A: Ziel war ein Redesign. Einzelne Teile sollten neu geschrieben werden. Es hat sich aber gezeigt, dass alles neu programmiert werden musste. Konzeptionelle Ideen konnten zwar übernommen werden, aber der Code ist vollständig neu. Das brachte vor allem eine starke Qualitätsverbesserung bei den Daten.

B: Höhere Datenqualität für den Benutzer. Beschleunigung der Abgleichzeiten.

D: Schnellere Online Abfragen für die Firmen.

? Wurde der Nutzen quantifiziert? Wenn ja, in welcher Form? Wenn nein, warum nicht?

A: Nein noch nicht. Es wird sich aber sicher in der Stundenabrechnung beim Support zeigen, dass der Supportaufwand in den nächsten Jahren stark zurückgehen wird. Hochrechnungen wurden vorgängig keine erstellt. Das neue Design wird auch den Aufwand beim Upgrade bei den rund achtzig Filialen stark reduzieren, da ein Upgrade nun automatisiert abläuft.

B: Beschleunigung der Abgleichzeiten (soll morgens vor 7.30 Uhr fertig sein).

Wie wurde das Projekt abgewickelt?

? Nach welcher Methode wurde vorgegangen (z.B. Extreme Programming)?

A: Phasenmodell nach Hermes. Die Prozesse innerhalb der Phasen waren mehr ad hoc organisiert.

B: Nach Hermes.

C: Das war am Anfang schwierig, vor allem wegen der unterschiedlichen Vorgehensweise auf Host und PC. Wir haben uns dann zwischen den beiden Welten vor allem mit klaren Namenskonventionen geholfen und so wurde das Projekt durchgezogen.

? Wie wurden die Projektmitarbeiter auf die verwendete Methode vorbereitet?

A: Die Mitarbeiter wurden durch mich vor jeder Phase wurden in einer kurzen Ausbildungssequenz über die Ziele und Inhalte orientiert.

B: Nein.

? Wie wurde geprüft, ob die Methode richtig angewendet wurde?

A: Eine projektnahe Qualitätssicherung fand nicht statt. Die Stelle Projektcontrolling beschränkt sich auf das Zusammenziehen von Statusberichten und führt keine detaillierten Kontrollen durch.

B: Gar nicht.

? Wurden spezielle Werkzeuge verwendet?

A: Keine.

B: Wäre mir nicht bekannt. Für ORACLE DB Arbeiten verwendeten wir die Oracle Tools.

C: Nur auf den PC. Auf den Host gibt es noch nichts dergleichen.

Grundsatz 1: Einbezug Benutzer

? Wurde dieser Grundsatz eingehalten? 4: C *5: B, D 6: A*

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Es gab einen Key-User einer Filiale, welcher schon bei den ersten Tests dabei war. Intern haben wir einen Produkteverantwortlichen, der auch für Schulung und First-Level Support bei anderen Filialen verantwortlich ist. Diese Person ist eigentlich auch ein reiner Benutzer, er hatte mit der Programmierung nichts zu tun. Wir haben uns jeweils auf für den Benutzer klar überprüfbare Punkte geeinigt.

B: Es gibt einen Produkteverantwortlichen im Team, der als Verbindungsperson zum Kunden diente.

C: Ist für mich noch schwierig zu beurteilen. Aus meiner Sicht hat man es versucht, aber am Anfang war es ein Problem.

? Wer hat die Spezifikation geschrieben?

A: Projektleiter. Es gibt eine grosse Kluft zwischen Informatikwissen und Fachwissen und wegen meiner früheren Tätigkeit, hatte ich die besten Kenntnisse in beiden Gebieten.

B: Projektleiter. Am Anfang war Produkteverantwortlicher und Projektleiter die gleiche Person. Später hat man das aufgetrennt.

C: In erster Linie durch den Projektleiter. Die jeweiligen Detailspezifikationen wurden von den Entwicklern erstellt.

D: Projektleiter.

? In welcher Form wurde die Spezifikation erstellt?

A: Schriftlich, Word Dokumente, gemäss Hermes.

B: Alles schriftlich (Word, Excel). Für diese Art von Produkt, wo man ein Redesign macht, gibt es eigentlich keine speziell geeigneten weitere Hilfsmittel oder Notationen.

C: Teilweise schriftlich, teilweise nur mündliche Absprachen. Am Anfang gab es da auch Missverständnisse aber mit den Konventionen, die im Laufe des Projektes eingeführt wurden, funktionierte es dann besser.

? Wurde die Spezifikation im Verlaufe des Projektes angepasst und wenn ja, warum?

A: Es fanden Anpassungen statt. Gewisse "nice to have" Komponenten wurden gestrichen.

B: Ja. Es gab Probleme bei der technischen Umsetzung, vor allem bei den Übertragungsgeschwindigkeiten von der Hostseite her. Änderungen wegen Benutzeranforderungen kamen nur sehr wenige, weil es ja primär um eine Verbesserung der Geschwindigkeit eines bestehenden Systems ging.

C: Die Spezifikationen wurden laufend angepasst. Aus verschiedenen Gründen, teils technische Probleme, teils auch dass Anforderungen von den Entwicklern falsch interpretiert wurden, weil sie das Fachwissen von der Benutzerseite nicht hatten.

? Wann erhielten die Anwender erste Programmversionen? Anteil Projektgesamtdauer: *80%*

B: Ab Januar 2000 Pilotversion.

? Wie häufig erhielten die Anwender neue Versionen?

A: Es gab etwa zehn Updates im produktiven System, wobei dies für den Anwender nicht sichtbar war, da es vor allem interne Anpassungen an der Software waren.

B: Es gab eine Pilotversion in einem Betrieb, dann wurden das System nach und nach in allen Betrieben eingeführt.

? Wie wurde überprüft, ob die Spezifikation eingehalten wurde?

A: Mit Hilfe von Tests durch den Benutzer. Diese Kontrolle war in diesem Projekt nicht sehr komplex.

B: Ausgewählte Benutzer haben mit der Software gearbeitet und haben noch nötige Änderungen schriftlich zu Handen der Entwicklung abgegeben.

C: Mit Tests, wo die einzelnen Teilprogramme zusammengehängt wurden.

Ich finde den Grundsatz "Einbezug Benutzer"
wichtig für das Gelingen eines Projektes: *7: A, B, C, D*

? Können Sie Ihre Gewichtung begründen ?

A: Betroffene sollen zu Beteiligten gemacht werden. Gerade bei einem Redesign eines bestehenden Systems müssen die Benutzer besonders gut einbezogen werden.

B: Es ist wichtig, dass die Benutzeranforderungen optimal abgedeckt werden.

C: Ich finde diesen Grundsatz sehr wichtig. Ein Vertreter der achtzig Firmen war immer involviert und konnte genau seine Bedürfnisse einbringen. Und das hat in diesem Projekt auch sehr gut funktionert.

D: Es ist wichtig, dass man in einem Projekt über sein eigenes Gebiet hinaus die Gesamtzusammenhänge begreift. Sonst kommt es schnell zu Missverständnissen.

? Haben Sie weitere Bemerkungen zum Grundsatz "Einbezug Benutzer"?

A: Ein System ist dann erfolgreich, wenn der Benutzer damit zufrieden ist und nicht wenn etwas technisch möglichst genial umgesetzt wurde. Zufriedenheit des Benutzers ist für ein Projekt immer auch das beste Marketing.

B: Häufig weiss der Benutzer am Anfang noch nicht genau was er braucht. Erst wenn er etwas sieht, kommt von ihm eine Reaktion. Prototyping einzusetzen ist deshalb ideal.

D: Nicht jeder muss über jedes Detail informiert sein, aber ein Gesamtüberblick sollte man haben.

Grundsatz 2: Einfach und klein

? Wurde dieser Grundsatz eingehalten? *4: A, B 7: D*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Dieser Grundsatz wurde nur teilweise eingehalten. Wegen der Fluktuation im Projektteam war es nicht immer ganz einfach.

B: Die personellen Ressourcen waren am Schluss in diesem Projekt sehr beschränkt, so dass die Teile für die einzelne Personen relativ umfangreich wurden.

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

C: Eine Aufteilung war hier zwingend, da Host und PC Technologien sehr unterschiedlich waren.

D: Unser Teil war so einfach wie möglich, bei den Programmen in Cobol und PL/SQL kann ich das nicht beurteilen.

? Enthält das Programm Funktionen, die nie gebraucht werden?

A: Keine.

B: Gibt es keine.

C: Sehr wenig. Teil des Redesigns war unnötige Funktionen zu entfernen.

D: Es gibt Funktionen, die im Moment nicht verwendet werden. Diese wurden vorbereitet um Änderungen einmal schneller integrieren zu können.

? Falls unötige Funktionalitäten vorhanden sind, warum?

D: Es war einfacher, diese Funktionen bereits jetzt zu erstellen, als erst zu einem späteren Zeitpunkt.

? Wieviel Schulung ist nötig, um mit dem Produkt zu arbeiten?

A: Für den Benutzer ca. ein Tag.

B: Für den Anwender braucht es nicht so viel Schulung. Die Einarbeitung für Entwickler ist schwieriger. Das war zu Beginn dieses Projektes auch ein Problem, wenn man in einem vorher unbekannten Bereich arbeiten muss. Das Vorgehen war häufig "try and error".

C: Auf Hostseite gibt es keine Schulung, da keine direkten Benutzerzugriffe stattfinden.

? Wie wurde die Arbeit unter den Projektmitarbeitern aufgeteilt?

A: Die Aufteilung erfolgte nach Fähigkeiten und Erfahrungen der Mitarbeiter.

B: Es wurde nach Bereichen Host / Pool DB =Schnittstelle Host-Oracle / DWH DB aufgeteilt.

C: Nach technischem Bereich Host / DB Server.

? Was waren die Gründe für die gewählte Aufteilung?

A: Schon aus Zeitgründen musste jeder dort eingesetzt werden, wo er am meisten bringt. Wir konnten uns in diesem Projekt nicht erlauben, Leute in neuen Gebieten einzusetzen und so gleichzeitig noch das Know How in der Firma auszubauen.

B: Komplexität des Projekts.

? Führte die gewählte Aufteilung zu Problemen? Wenn ja, welche?

A: Es gab Probleme, weil die Leute vor allem in ihrem Bereich zusätzliche Erfahrungen machten und die Verständigungslücke zwischen den Gebieten wurde eher grösser als kleiner. Vor allem zwischen Hostwelt und Open Systems. Es fehlte die gemeinsame Sprache.

B: Es gab Probleme immer dann, wenn fachübergreifende Diskussionen nötig waren.

C: Es gab Probleme, weil jeder technische Bereich seine ganz spezifischen Probleme hatte, die von den anderen Leuten nicht immer verstanden und ernst genommen wurden.

? Wieviele Projektmitglieder kennen den gesamten Sourcecode?

A: Vermutlich nur ich, da ich als einziger in allen Bereichen mindestens involviert war. Nicht dass ich alles im Detail beherrsche, aber ich würde mich in allen Teilsystemen zurecht finden.

B: Jeder kennt seinen Teil und ein bisschen, was der andere macht. Die Gesamtheit ist eher eine Graybox , aber der einzelne Fachbereich ist eine Blackbox. Fachbereiche Host und Server.

C: Sicher der Projektleiter. Weiter sind es insgesamt etwa vier Leute die zusammen allen Source kennen (Host / DB).

Ich finde den Grundsatz "Einfach und klein"
wichtig für das Gelingen eines Projektes: *4: A 5: B, C, D*

? Können Sie Ihre Gewichtung begründen ?

A: Ein Projekt muss sicher strukturiert werden, aber es darf nicht aufgetrennt werden. Projekte mit bis zu zwanzig Personen sollte immer versucht werden, dass möglichst alle Projektmitarbeiter eine Gesamtsicht haben. Am Besten würden immer alle im selben Büro arbeiten, auch wenn man unterschiedliche Arbeiten erledigt.

B: Wenn es möglich ist ja, die Realität sieht anders auch. Auch gerade dieses Projekt, dass sehr fächerübergreifend aufgebaut ist, ist unter Berücksichtigung aller Details nicht "einfach und klein".

C: Es ist wichtig. Ist aber gerade auf dem Host je nach Benutzeranforderungen, sehr schwierig zu erfüllen. Schlussendlich sind die Benutzeranforderung das klar höher zu gewichtende Kriterium.

Grundsatz 3: Qualität in allen Teilen

? Wurde dieser Grundsatz eingehalten? *3: A 4: C 6: B, D*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Es fand keine projektnahe Qualitätskontrolle statt. Auch die Produktekontrolle war minimal. Es fanden zwar Benutzertests statt, aber die Qualitätssicherung wurde nicht planmässig geprüft. Es fehlte hier an der Erfahrung auf allen Stufen von Auftraggeber über Projektleiter und Entwickler.

B: Die Qualität war ein wichtiges Ziel in diesem Projekt und wurde auch erreicht. Das war auch der Hauptgrund, warum die Termine nicht eingehalten werden konnten.

C: Hier gab es Probleme. Im Host Umfeld arbeiten wir mit sehr starren Regeln. Wir waren bei allen Änderungen auch immer die Langsamsten und aus Zeitnot wurde dann häufig erstmal irgend ein "Klimmzug" gemacht. Wobei jetzt im nachhinein konnte vieles davon korrigiert werden.

D: Ich denke das Projekt wurde gut geplant. Wie sauber die Dinge programmiert sind, kann ich nur schwer beurteilen, aber ich schätze die Beteiligten Entwickler als sehr kompetent in ihrem Gebiet ein.

? Gibt es im Endprodukt bekannte, nicht behobene Fehler?

A: Nein, höchstens Kleinigkeiten welche nicht als eigentliche Fehler angeschaut werden können.

B: Fehler wurden fortlaufend behoben.

C: Bis jetzt sind keine nenneswerte Fehler aufgetaucht. Das System läuft stabil.

? Wurden regelmässig Tests durchgeführt?

A, C, D: Ja.

B: Tests werden vom Produkteverantwortlichen laufend (täglich) durchgeführt.

? In welcher Form wurde getestet?

A: Intuitiv, ohne konkrete Planung. Es wurde an Beispielen Geschwindigkeit und Datenintegrität überprüft.

B: Bestimmte Testfälle werden ausgesucht, getestet und dann manuell überprüft und Fehler direkt behoben.

C: Hostseitig gab es verschiedene Arten von Tests: Einzeltest jedes Programmes, dann Test zusammen mit der Datenbank. Anschliessend gingen die von uns generierten Datenfiles zu den Oracle Leuten, welche Tests machten, ob der Import funktioniert. Als nächstes wurde das ganze mit einer grösseren Datenmenge getestet und schlussendlich gab es ein Test mit allen Daten. Der Aufwand für die Tests über etwa vier bis fünf Schnittstellen war sehr aufwendig.

D: Die Tests liess man auf einem Testsystem laufen und verifizierte die Resultate manuell.

? Gibt es Teile in der Software die neu geschrieben werden sollten?

A: Ja.

B: Nein. gibt es im Moment keine. Entfernung von Redundanz und gute Qualität war genau eines der Ziele in diesem Projekt. Eine Erweiterung ist geplant=neues Projekt.

C: Das gibt es immer.

? Warum sollten welche Teile neu geschrieben werden?

A: Nicht wegen Fehlern sondern wegen erweiterten Benutzeranforderungen.

B: Nur Erweiterung im Rahmen des neuen Projekts.

C: Hostseitig führen wir eine Pendenzenliste für nötige Anpassungen. Das ganze ist ein laufender Prozess, auch mit neuen und geänderten Benutzeranforderungen.

? Gab es eine institutionalisierte Form für Design- und Codereview?

A: Gab es nicht.

B: Vor allem Selbstkontrolle. Es wurden eigene Standards und Konventionen definiert und jeder hat sich an diese gehalten.

C: Nein, nicht auf Stufe Sourcecode. Es wurde einfach nach den vorhandenen internen Richtlinien entwickelt.

? Warum fanden keine Reviews statt?

A: Seitens der Firma gibt es hier keine Vorlagen, bei uns fehlte die Erfahrung und die Zeit, so etwas zu entwickeln und einzuführen.

? Ist die Software hinreichend dokumentiert?

A: Für meine Qualitätsvorstellungen ist die Dokumentation nicht ausreichend. Sie ist sicher hinreichend, für jemand, der sich neu in das System einarbeiten müsste. Es fehlt aber eine Dokumentation über gemachte Erfahrungen in diesem Projekt, etwas, dass gerade für zukünftige Projekte sinnvoll wäre.

B, C: Ja.

? Welche Art von Dokumentationen wurden erstellt?

A: Sourcecode Dokumentation und Beschreibung von Programmabläufen.

B: Systementwicklungsdokumentation nach Hermes und ein Benutzerhandbuch.

C: Es gibt für jedes Programm eine einfache Dokumentation für die Leute in der Produktion und eine Detaildokumentation für die Entwicklung.

Ich finde den Grundsatz "Qualität in allen Teilen" wichtig für das Gelingen eines Projektes: *6: A, B 7: C, D*

? Können Sie Ihre Gewichtung begründen ?

A: Nachhaltigkeit wird nur mit qualitativ guter Arbeit erreicht.

B: Qualität ist für mich immer ein wichtiges Ziel in einem Projekt.

C: Qualität ist für mich persönlich sehr wichtig, kommt direkt nach den Benutzeranforderungen. Im ersten Moment ist das sehr aufwendig und kostet Geld, aber mit der Zeit spart man sich viel Ärger.

D: Qualität erachte ich es als eines der wichtigsten Kriterien, vor allem auch gegenüber dem Kunden.

? Haben Sie weitere Bemerkungen zum Grundsatz "Qualität in allen Teilen"?

A: Es braucht auch klare Kriterien, wie die Qualität überprüft wird. Zudem sollten die Erfahrungen in jedem Projekt ausgewertet werden, um davon bei weiteren Projekten zu profitieren.

D: Bei vielen Projekten wird auf Kosten der Einhaltung von Terminen bei der Qualität Abstriche gemacht, und ich erachte das als falsch.

Grundsatz 4: Zeitplan und Ziele

? Wurde dieser Grundsatz eingehalten? *2: A, B, C, D*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Mir fehlte die Erfahrung als Projektleiter, es fehlte aber auch die Unterstützung für diese Arbeit innerhalb der Firma. Zudem war diese Projektleitung eine Zusatzaufgabe zu allen anderen Arbeiten.

B: Weil wir sehr viel Wert auf die gute Qualität legten, konnte der ursprüngliche Zeitplan nicht eingehalten werden.

C: Das Projekt lief anfangs nicht sehr strukturiert, ein wenig führungslos. Zeitlich war es zwar schon organisiert, aber die Ziele waren zu wenig klar kommuniziert. Im Laufe des Projekts wurde das korrigiert.

D: Es fehlte die Erfahrung mit den verwendeten neuen Technologien.

? Wie wurde der Zeitplan festgelegt?

A: Mündlich und anschliessend schriftlich in Sitzungsprotokollen festgehalten.

B: Projektplan mit MS Project erstellt.

C: Der existierte mal schriftlich, aber wegen grossen technischen Problemen stimmte der bald nicht mehr.

D: Schriftlich, aufgeteilt in die Phasen Planung - Implementierung - Tests, relativ grob.

? Wie detailliert war der Zeitplan aufgeteilt?

A: Aufgeteilt nach einzelnen Teilprogrammen in der Grössenordnung von Wochen.

B: Der Zeitplan war insofern nicht sehr massgebend, da nach der Analysephase einige Personen die Firma verlassen haben und dann die für das Projekt nötigen Resourcen fehlten.

? Waren jedem Mitarbeiter der Zeitplan / Ziele bekannt?

A: Ja, der Zeitplan wurde an den wöchentlichen Meetings besprochen.

B: Der Zeitplan war bekannt.

C: Ja, der Zeitplan war mehr oder weniger immer bekannt.

D: Ja, war allen bekannt.

? Wie wurde die Einhaltung Ziele / Zeitplan überprüft?

A: Wöchentlich an den Projektmeetings.

B: Wegen den Abgängen von Personen war eine laufende Kontrolle des Zeitplanes schwierig.

? Wie wurden auf Abweichungen Realität <-> Zeitplan reagiert?

A: Der Zeitplan wurde der Realität angepasst. Ich versuchte den Druck von Seiten des Auftraggebers nicht auf die Mitarbeiter weiterzugeben. Es gab keine Überstunden wegen dieses Projektes. Deshalb hat sich der Endtermin ziemlich hinausgezögert.

B: Ein Projektstop wurde einmal diskutiert aber man das Projekt dann doch fertiggestellt. Es wurde einfach der Endtermin hinausgeschoben.

C: Es gab einen Projektstop und es wurden erst mal technische Abklärungen der Machbarkeit gemacht. Schlussendlich wurde einfach der Endtermin hinausgeschoben.

D: Projektende wurde verschoben.

Ich finde den Grundsatz "Zeitplan"
wichtig für das Gelingen eines Projektes: *5: A, B, C 7: A*

? Können Sie Ihre Gewichtung begründen ?

A: Ohne konsequente Planung lässt sich kein Projekt durchführen. Auch wenn diese Planung laufend angepasst werden muss, wichtig ist einfach, dass man eine hat. Der Detailierungsgrad muss aber sicher nicht tagesweise sein, wochenweise ist sinnvoller ausser vielleicht bei Systemwechseln wo fixe Tage festgelegt werden müssen.

B: Ich würde eher von Zeitrahmen sprechen als von Zeitplan.

C: Dieser Grundsatz wäre wünschenswert. Aber meistens werden bei der Analyse die Details vernachlässigt. Und Details können plötzlich zu grossen Problemen werden. Ich habe kein Projekt erlebt, wo Zeitpläne wirklich gut eingehalten werden konnten, ausser es wurden enorme Zeitreserven von Anfang an eingeplant. Zeitpläne sind fast immer zu optimistisch aufgestellt.

D: Eine schlechte Planung eines Projektes führt häufig zu negativer Entwicklung bei der Qualität.

? Haben Sie weitere Bemerkungen zum Grundsatz "Zeitplan"?

A: Ohne Erfahrung ist eine gewisse Gefahr, dass man zu detaillierte Zeitpläne aufstellt. Der Wartungs- und Anpassungsaufwand für diese Pläne wird dann zu gross und man aktualisiert sie nicht mehr. Deshalb ein möglichst einfachen Zeitplan, diesen dafür immer à jour behalten.

B: Ein Zeitplan nützt nur dann etwas, wenn er realistisch gehalten wird.

C: Hosts sind riesige Gebilde und es stellen sich vor allem viele Mengenprobleme. In einem Rechenzentrum laufen viele verschiedene Programme mit klaren Prioritäten. Es fehlen dort die Flexibilitäten um auf ständig änderende Zeitpläne zu reagieren. Auch haben auf dem Host einzelne Programme sehr starre Abhängigkeiten voneinander.

Grundsatz 5: Spass der Beteiligten

? Wurde dieser Grundsatz eingehalten? *2: C 4: A 5: B 6: D*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Die hohen Fluktuationsrate, der Beizug von externen Mitarbeitern gab gewisse Probleme.

B: Wenn mehrere Leute zusammenarbeiten können Probleme auftreten wegen unterschiedlichen Auffassungen betreffend Durchführung.

C: Gerade in diesem Projekt hat die Kommunikation zwischen verschiedenen Fachleuten zeitweise überhaupt nicht gespielt. Es gab eine Phase, wo Lösungen und Ideen diskutiert wurden, zu denen ich von meinem Berufsverständnis her nicht mehr ja sagen konnte. Vor allem der Umgangston war in dieser Phase schlecht.

D: Man hatte genügend Freiheiten bei der Arbeit. Man konnte gegenseitig voneinander lernen

? Was war Ihre wichtigste Motivation bei der Arbeit an diesem Projekt? Warum?

A: Das alte fehlerhafte System zu beheben durch ein besseres Produkt. Erfahrungen als Projektleiter sammeln zu können. Da es auch viele schwierigen und negativen Erfahrungen gab, war das Projekt umso lernreicher.

B: Ein qualitativ gutes Produkt zu erstellen. Vor allem als sich einmal ein Weg zu einer Lösung abzeichnete war das motivierend. Auch dass nicht immer alles nach Plan läuft, war eine Herausforderung.

C: Für mich als Entwickler war es in Bezug auf die verschiedenen Schnittstellen eine grosse Herausforderung und meine Motivation war, diese Probleme zu lösen.

D: Ich lernte Neues in Gebieten, die über meine übliche Tätigkeit hinausgeht.

? Gab es für sie persönlich wichtige Ziele, die nicht erreicht wurden? Welche? Warum?

B: Keine.

C: Fachlich wurden alle Ziele erreicht, zwischenmenschlich blieben Probleme ungelöst.

D: Keine.

? Was hat Ihnen bei diesem Projekt am wenigsten gefallen? Warum?

A: Die Kritik von aussen. Es gab relativ viel Kritik, vermutlich weil es ein relativ wichtiges System für die Firma ist. Vermutlich ist das einfach so, dass die wichtigsten Projekte, die am meisten Leute betreffen, auch am meisten kritisiert werden. Weiter störte mich die Rolle, die der Auftraggeber spielte; ihm fehlte schlicht auch die Erfahrung, so dass er mich nur schlecht unterstützen konnte. Diese Zusammenarbeit ergab fast mehr Probleme als Lösungen.

B: Der Zeitplan war nicht realistisch. Man ging von falschen Annahmen aus.

C: Der Umgangston mit Personen im Projekt, welche eine absolut intolerante Haltung an den Tag legten.

D: Ich war nicht von Anfang an beim Projekt dabei. Ich hätte früher auf gewisse Probleme aufmerksam machen können und man hätte eher darauf reagiert.

? Was hat bei diesem Projekt besonders gut funktioniert? Warum?

A: Das Kernteam, also die Leute, die während der ganzen Projektdauer dabei waren, hat unter sich sehr gut funktioniert. Es hätte auch niemand wegen dieses Projektes gekündigt. Und dann gefiel mir, dass wir schlussendlich doch erfolgreich abschliessen konnten, trotz des zu späten Termins.

B: Das Lösen der Teilprobleme des jeweiligen Fachbereichs für den der einzelne Projektmitarbeiter zuständig ist.

C: Das es eine technisch sehr anspruchsvolle Aufgabe war, die Probleme zu lösen.

D: Mit wenigen Ausnahmen war die Kommunikation innerhalb des Teams sehr gut.

? Wenn Sie rückblickend Ihre Rolle im Projekt ändern könnten, was würden Sie ändern?

A: Ich würde viel mehr steuern und viel weniger selber machen. Weil es mich interessierte und mir auch Spass macht, habe ich mich zu häufig mit den Entwicklern zusammen mit systemtechnischen Problemen auseinandergesetzt. Davon muss ich mich in Zukunft lösen, um mich mehr auf die Mangementaufgabe zu konzentrieren. Das heisst auch, Entscheide und Verantwortung vermehrt nach unten delegieren. Auch würde ich in Zukunft viel stärker darauf pochen, dass auch der Auftraggeber seine Pflicht wahrnimmt.

B: Eigentlich nicht. Mit den Erfahrungen im nachhinein ist man immer klüger.

C: Ja, ich würde mir wünschen, dass es eine Einführungsphase geben würde, wo man sehen würde, ob die Projektmitarbeiter vom Wissensstand und menschlich zusammen passen.

D: Nichts.

? Gab es im Laufe des Projektes Konflikte unter den beteiligten Personen?

A: Es gab Konflikte, vor allem zwischen Leuten auf der Hostseite und den DB-Entwicklern. Gerade die Hostentwickler leben sehr stark in ihrer Welt und haben nur sehr wenig Ahnung, was in anderen Gebieten läuft. Die Konflikte erreichten den Höhepunkt mit den externen Mitarbeitern, deren Ziel nicht primär ein gutes Projekt war, sondern ein dauerhaftes Engagement für sich bei unserer Firma und die Vermehrung ihres persönlichen Wissens. Teilweise versuchten sie Ideen und bereits beschlossene Lösungen wieder über den Haufen zu werfen, nur weil es ihnen nicht gepasst hat.

B: Ja, die gab es. Es waren vor allem Verständigungsprobleme zwischen den einzelnen Fachbereichen.

C: Wie schon erwähnt, die gab es.

D: Es gab Probleme mit externen Mitarbeitern. Ich war davon wenig betroffen.

? Wie wurden aufgetretene Konflikte gelöst?

A: MIt Gesprächen unter vier Augen, fast alles lief über mich als Projektleiter. Es waren kleine fachliche Probleme, aber durch die Häufigkeit und die Art des Verhaltens von gewissen Personen wurden es grosse Probleme zwischenmenschlicher Art.

B: Der Projektleiter hat vermittelt zwischen den Personen.

C: Die Konflikte wurden eigentlich nie völlig gelöst, man hat dann einfach die Personen, die sich nicht verstanden haben, möglichst voneinander getrennt.

D: Die Verträge mit den externen Personen wurden nicht verlängert.

? Gab es personelle Wechsel im Projektteam? Gründe?

A: Gab es, sehr viele. Es waren total 15 Personen beteiligt, das Projektteam bestand aber nie aus mehr als sieben Personen, also ziemlich viele Wechsel. Dazu kamen noch die externen Mitarbeiter.

B: Es gab Abgänge, welche nicht primär in Zusammenhang mit dem Projekt standen.

D: Abgang der externen Mitarbeiter.

? Gab es gesellschaftliche Anlässe innerhalb des Projektteams? Welche? Von wem initiert?

A: Gab es leider keine. Das wurde auch von Seite Auftraggeber tendenziell unterbunden. Auch wegen den vielen Wechseln im Projektteam und den vielen technischen Probleme, die zu lösen waren, kamen solche Anlässe nicht zustande.

B: Nein, gab es keine.

C: Nein. Für mich wäre so etwas enorm wichtig und ich habe das hier vermisst. Es gab nur gerade fachliche Diskussionen, aber nie zum Beispiel ein gemeinsames Nachtessen oder etwas in dieser Richtung.

D: Gab es keine, leider.

Ich finde den Grundsatz "Spass der Beteiligten"
wichtig für das Gelingen eines Projektes: *6: B, C 7: A, D*

? Können Sie Ihre Gewichtung begründen ?

A: Mit Spass ist man motiviert oder umgekehrt und das führt schlussendlich zum Erfolg. Und gemeinsam ein Ziel zu erreichen ist wiederum sehr motivierend für alle Mitarbeiter. Solche Entwicklungen wirken sich dann auch wieder auf die Kultur innerhalb der Firma aus.

B: Es ist einfach so, dass wenn die Mitarbeiter Spass haben, sind sie motiviertert und dass kann sich zum Beispiel auch auf Einhaltung des Zeitplans wieder positiv auswirken.

C: Jedes Projekt ist persönliches Engagement und Teil des Lebens eines jeden einzelnen Beteiligten. Am Schluss hat man gemeinsam etwas erreicht und dem sollte in irgend einer Form Rechnung getragen werden.

D: Arbeiten nach engen Vorgaben und schlechte Kommunikation verhindern Freude an der Arbeit. Wer kein Spass hat, denkt auch weniger mit im Projekt.

Fragebogen / Interview Projekt 6

Allgemeine Angaben zum Projekt

Ihre Rolle im Projekt:	*A: Projektleiter B: Entwickler*		
Fachgebiet:	*Business Software*		
Zeitdauer des Projektes:	Beginn am:..... *1997*	Ende am: *2000*	
Anzahl beteiligte Personen:	Total:..... 7-10	davon Entwickler: 4-5	
Aufwand in Arbeitstagen:	Total:..... *4000*	davon Entwicklung:.....*3000*	
Umfang des Projektes:	Anzahl Applikationen:.....*7*		
Anzahl Zeilen Sourcecode:	dieses Projekt: *103K*	eigene Library: *...86K*	Drittlibraries:*.... 163K*
Anzahl Klassen:	dieses Projekt: ...*160*	eigene Library: ... *335*	Drittlibraries:.....*473*
Programmiersprache(n):	*Object Pascal, PL-SQL*		
Entwicklungsumgebung(en):	*Borland Delphi 1,2,4 / Oracle*		

War das Projekt erfolgreich?

Termine wurden eingehalten: *6: A, B*

Kostenrahmen wurde eingehalten:
A: Keine Aussage möglich. Der Kostenrahmen wurde praktisch nicht überwacht.

Software ist fehlerfrei: *5: A, B*

Software erfüllt die Benutzeranforderungen: *6: A, B*

Software ist wartbar: *4: A, B*

? Welche Rahmenbedingungen machten das Projekt speziell einfach?

A: Gutes Verhältnis zwischen Projektmitarbeitern und Anwendern (teilweise langjährige Arbeitsbeziehungen)
Guter Support Informatik-Leitung für das Projekt / Projektteam
Rückhalt bei Geschäftsleitungsmitglied aus Benutzerbereich
Tolles Team
B: Software wurde in der Firma verwendet.

? Welche Rahmenbedingungen machten das Projekt speziell kompliziert?

A: Lange Durchlaufzeit des Gesamtprojekts. Parallel dazu laufende Reorganisation der Anwender (Aufbau- und Ablauforganisation, Verlagerung von Aufgaben).
B: Siehe oben / viele unterschiedeliche Interessen (fachlich) mussten unter einen Hut gebracht werden.

? Was ist der Nutzen der erstellten Software?

A,B: Effizientere Bearbeitung von Schadenfällen (Versicherung).

? Wurde der Nutzen quantifiziert? Wenn ja, in welcher Form? Wenn nein, warum nicht?

A: Kosten-/Nutzenanalyse, Nachweis Stellenabbau im Schadenbereich.
B: Kosten-/Nutzenanalyse.

Wie wurde das Projekt abgewickelt?

? Nach welcher Methode wurde vorgegangen (z.B. Extreme Programming)?

A,B: IFA (analog Hermes).

? Wie wurden die Projektmitarbeiter auf die verwendete Methode vorbereitet?

A,B: Schulung IFA (1 Tag).

? Wie wurde geprüft, ob die Methode richtig angewendet wurde?

A,B: Ergebnisprüfung durch den Projektleiter.

? Wurden spezielle Werkzeuge verwendet?

A,B: ERWIN für DB-Design / Visio.

Grundsatz 1: Einbezug Benutzer

? Wurde dieser Grundsatz eingehalten? *6: A, B*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Nicht homogene Benutzergruppen, es konnten nicht alle Benutzergruppen Vertreter im Projektteam platzieren (z.B. wegen geografischer Verteilung, Arbeitsbelastung)

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Viele Punkte des Grundsatzes wurden eingehalten (Prototyping, visuelle Darstellungen, Key-User, Sammeln und Bewerten der Benutzeranforderung durch den Key-User).

B: Im Grundsatz aufgeführte Punkte wurden eingehalten.

? Wer hat die Spezifikation geschrieben?

B: Projektleiter, Entwickler.

B:Anwender, Projektleiter, Entwickler.

? In welcher Form wurde die Spezifikation erstellt?

A, B: Worddokumet / Visio.

? Wurde die Spezifikation im Verlaufe des Projektes angepasst und wenn ja, warum?

A: Spez. mussten aufgrund gesetzlicher und applikatorischer Anforderungen angepasst werden.

B: Spez. mussten aufgrund gesetzlicher Anforderungen angepasst werden. Spez. mussten aufgrund von Veränderungen im Geschäftsablauf angepasst werden.

? Wann erhielten die Anwender erste Programmversionen? Anteil Projektgesamtdauer A,*B: 50%*

? Wie häufig erhielten die Anwender neue Versionen?

A, B: Zu Beginnn wöchentlich danach monatlich und zuletzt alle 3-4 Monate.

? Wie wurde überprüft, ob die Spezifikation eingehalten wurde?

A, B: Durch Fachvertreter und Anwender.

Ich finde den Grundsatz "Einbezug Benutzer"
wichtig für das Gelingen eines Projektes: *6: A 7: B*

? Können Sie Ihre Gewichtung begründen ?

A: Projekt ist letztendlich nicht für die Informatik, sondern für den Anwender / Kunden.

B: Der Kunde ist König!!

? Haben Sie weitere Bemerkungen zum Grundsatz "Einbezug Benutzer"?

A: "DEN" Benutzer gibt es oft nicht: verschiedene Interessen bewirken, dass die Benutzer eine nicht homogene Gruppe bilden mit unterschiedlichen und oft gegenläufigen Anforderungen.

Projektmanagement-Wissen ist bei Benutzern nicht vorhanden, oft auch wenig Verständnis für die Vorgehensweise und den Zeit- und Ressourcenbedarf für ein Projekt.

Angst vor Neuem lässt viele Anwender Abstand von einem Projektengagement nehmen.

Benutzer, die sich intensiv in einem längeren Projekt engagieren wollen, verlieren ihren Platz in der Linie, was zu Problemen führen kann.

Grundsatz 2: Einfach und klein

? Wurde dieser Grundsatz eingehalten? *4: A 7: B*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Das Gesamtprojekt war umfangreich: Viele Anforderungen, weil teilweise 20-jährige Anwendungen abgelöst werden mussten, führten zu einer nicht optimalen Grösse.

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Es arbeiteten nie mehr als 2-3 Personen je Modul. Tranchierung: Ausbauschritte wurden mit dem Auftraggeber festgelegt und bezüglich Funktionsumfang und Terminen definiert.

B: Es arbeiteten nie mehr als 2-3 Personen je Modul.

? Enthält das Programm Funktionen, die nie gebraucht werden?

A: Ja, wurden aber wieder ausgebaut (1 Fall).

B: Ja.

? Falls unötige Funktionalitäten vorhanden sind, warum?

A, B: Falsche Spez. und Informationen durch Fachvertretung.

? Wieviel Schulung ist nötig, um mit dem Produkt zu arbeiten?

A, B: 2 Tage.

? Wie wurde die Arbeit unter den Projektmitarbeitern aufgeteilt?

A: Projektleitung / Spezifikation / Design / DB / Client-Teil / Schnittstellen / Host: je nach Fähigkeiten der Projektteam-Mitglieder.

B: Sezifikation / Design / DB / Client-Teil / Schnittstellen / Host.

? Was waren die Gründe für die gewählte Aufteilung?

A, B: Kleine überschaubare Module, gegenseitige Kontrolle an den Schnittstellen zwischen den Modulen.

? Führte die gewählte Aufteilung zu Problemen? Wenn ja, welche?

A: Manchmal (eher selten) unterschiedliche Präferenzen im Team: Termindruck beim Projektleiter, Wunsch nach "perfekten" Lösungen bei Entwicklern.

B: Nein.

? Wieviele Projektmitglieder kennen den gesamten Sourcecode?

A, B: Keiner.

Ich finde den Grundsatz "Einfach und klein"
wichtig für das Gelingen eines Projektes: *6: A 7: B*

? Können Sie Ihre Gewichtung begründen ?

A: Einfach und Klein macht es möglich, in absehbarer Zeit Lösungen zu bringen - vielleicht noch nicht vollständige Lösungen, aber hilfreiche, die dann in einer späteren Phase ausgebaut und perfektioniert werden können.

B: Überschaubarkeit der Module.

Grundsatz 3: Qualität in allen Teilen

? Wurde dieser Grundsatz eingehalten? *4: A 5: B*

? Aus welchen Gründen wurde von diesem Grundsatz abgewichen?

A: Zum Teil wurde die Dokumentation nicht sauber nachgeführt oder erstellt, weil dem Entwickeln von Ausbauschritten höhere Priorität eingeräumt wurde.

B: Zum Teil wurde die Dokumentation nicht sauber nachgeführt oder erstellt.

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A, B: sorgfältige Planung, gutes Design, Rücksprache mit den Fachvertretern.

? Gibt es im Endprodukt bekannte, nicht behobene Fehler?

A, B: Nein.

? Wurden regelmässig Tests durchgeführt?

A, B: Ja.

? In welcher Form wurde getestet?

A: Einzeltest, Kettentest, Benutzertests mit Protokollen. Gegenseitige Tests unter Entwickler, Test durch Projektleiter, Test durch Fachvertreter.

B: Gegenseitige Tests unter Entwickler, Test durch Projektleiter, Test durch Fachvertreter.

? Gibt es Teile in der Software die neu geschrieben werden sollten?

A, B: Ja.

? Warum sollten welche Teile neu geschrieben werden?

A, B: Kunde kannte die Abläufe in der eigenen Abteilung zu wenig und hat die Bedürfnisse falsch eingebracht.

? Gab es eine institutionalisierte Form für Design- und Codereview?

A: Ja (aber informell, ein Mitglied des Teams hat diese Aufgabe übernommen).

B: Ja/Nein.

? In welcher Form wurden die Reviews durchgeführt?

A, B: Review durch das Team und die Integrationsstelle.

? Ist die Software hinreichend dokumentiert?

A, B: Nein.

? Welche Art von Dokumentationen wurden erstellt?

A, B: Word / Kommentar in Code / Helpfile.

Ich finde den Grundsatz "Qualität in allen Teilen"
wichtig für das Gelingen eines Projektes: *7: A, B*

? Können Sie Ihre Gewichtung begründen ?

A: Nur durch Qualität kann die Ressourcenplanung eingehalten werden. Sie garantiert auch, dass nach Projektabschluss nicht noch hohe Wartungsaufwände für nicht behobene Mängel anfallen, welche nicht mehr sauber durch das Projektcontrolling gesteuert werden können.

B: Nur durch Qualität kann die Ressourcenplanung eingehalten werden.

Grundsatz 4: Zeitplan

? Wurde dieser Grundsatz eingehalten? *6: A, B*

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Fixe Planung, formelle Absprache zwischen Steuerungsausschuss / Projektcontrolling / Benutzer-Projektleiter und Informatik, wöchentliche Koordinationssitzung.

B: Wöchentliche Koordinationssitzung.

? Wie wurde der Zeitplan festgelegt?

A, B: schriftlich und mündlich.

? Wie detailliert war der Zeitplan aufgeteilt?

A, B: Tage / Wochen.

? Waren jedem Mitarbeiter der Zeitplan / Ziele bekannt?

A, B: Ja.

? Wie wurde die Einhaltung Ziele / Zeitplan überprüft?

A, B: Koordinationssitzung / Projektleiter-Sitzung.

? Wie wurden auf Abweichungen Realität <-> Zeitplan reagiert?

A: Zielanpassung, Anpassung Zeitplan, Ressourcen umverteilen, fallweise auch Terminverschiebung.

B: Zielanpassung, Anpassung Zeitplan, Ressourcen abstellen.

Ich finde den Grundsatz "Zeitplan"
wichtig für das Gelingen eines Projektes: *7: A, B*

? Können Sie Ihre Gewichtung begründen ?

A: Realistische und damit einhaltbare Zeitpläne schaffen Vertrauen zwischen Projekt und Anwendern Sie lassen den Projektmitarbeitern die Chance, ihre Ziele und damit Arbeitszufriedenheit zu erreichen. Ein realistischer Zeitplan enthält auch versteckte Termin-Reserven, so dass bei einer Verzögerung nicht das ganze Projekt oder der Gesamtplan zur Frage steht (Sicherheit, Stabilität).

B: Nur über einen realistischen Zeitplan bleibt das Arbeitsklima konstant gut. Durch einen extrem engen Terminplan leidet die Qualität des Produktes und das Arbeitsklima.

Grundsatz 5: Spass der Beteiligten

? Wurde dieser Grundsatz eingehalten? *5: A 7: B*

? Woraus schliessen Sie, dass dieser Grundsatz eingehalten wurde?

A: Neue Büroräumlichkeiten, super Team, gutes Umfeld, Team wurde von anderen Teams auch als Beispiel angesehen.

B: Neue Büroräumlichkeiten, super Team, super Teamleiter.

? Was war Ihre wichtigste Motivation bei der Arbeit an diesem Projekt? Warum?

A: Erfolgsaussichten, gute Mitarbeiter, Rückhalt bei verschiedenen Stellen, positives Feedback.

B: Interessantes Projekt, super Team, super Teamleiter.

? Gab es für sie persönlich wichtige Ziele, die nicht erreicht wurden? Welche? Warum?

A, B: Nein.

? Was hat bei diesem Projekt besonders gut funktioniert? Warum?

A: Gute, offene Kommunikation unter den Mitgliedern. Zielerreichung.

B: Team -> gute, offene Kommunikation unter den Mitgliedern.

? Wenn Sie rückblickend Ihre Rolle im Projekt ändern könnten, was würden Sie ändern?

A, B: Nichts.

? Gab es im Laufe des Projektes Konflikte unter den beteiligten Personen?

A: Zwei ungeeignete Fachvertreter.

B: Ja, ein Fachvertreter torpedierte das Projekt aus persönlichen Gründen.

? Wie wurden aufgetretene Konflikte gelöst?

A, B: Fachvertreter wurde ausgetauscht.

? Gab es personelle Wechsel im Projektteam? Gründe ?

A, B: Fachvertreter - war nicht geeignet für Projektarbeit.

? Gab es gesellschaftliche Anlässe innerhalb des Projektteams? Welche? Von wem initiert?

A, B: Ja, div. Feste bei erreichen von Meilensteinen, Informatikabteilung.

? Nahmen Sie an solchen Anlässen teil? Warum? Warum nicht?

A: Ja, wegen Funktion (Projektleiter) aber mehr noch aus persönlichem Interesse.

B: Ja, gehört dazu.

Ich finde den Grundsatz "Spass der Beteiligten"
wichtig für das Gelingen eines Projektes: *7: A, B*

? Können Sie Ihre Gewichtung begründen ?

A: Glückliche Hühner legen grosse Eier! Ein motiviertes Team versetzt Berge und unterstützt das Projekt in allen Phasen und Lagen.

B: Nur durch ein motiviertes Team können die Qualität des Produktes und die Termine eingehalten werden.