

# Partizipation im Internet

## Projektplanung

Marco Rademacher

7. November 2003

### Zusammenfassung

In dieser Vorlesung sollen die Grundlagen hinsichtlich der Planung eines Software-Entwicklungsprozesses gelegt werden. Dazu werden verschiedene Prozessmodelle vorgestellt. Am Ende sollen die Teilnehmer selbst festlegen, wie sie ihren Software-Entwicklungsprozess gestalten wollen, und ihn mit der Semesterplanung verknüpfen.

## 1 Projektarbeit

Kleinere Probleme des Alltags lassen sich sofort bewältigen und sind damit gelöst. Wird das Problem komplexer, braucht es zur Lösung schon längere Zeit, und man sieht sich hier oftmals dem ersten großen Handicap konfrontiert, wenn das Problem innerhalb einer Frist gelöst werden muss. Wird das Problem so komplex, dass man auf die Hilfe anderer angewiesen ist, behält man meist die Fäden in der Hand, indem man sich mit Teilproblemen an Spezialisten wendet. Wird aber die gesamte Problemlösung einer Gruppe übertragen, so müssen sich die Mitglieder untereinander abstimmen, damit ein gemeinsames Produkt entsteht.

Grundsätzlich entsteht das Produkt erst durch die Arbeit daran. Die Abstimmung zwischen den Arbeitenden ist letztlich eine Zeit, in der nicht am Produkt selbst weitergearbeitet wird. Bei komplexeren Projekten wird man um Abstimmungsarbeit nicht herumkommen, muss aber im Sinne produktiver Arbeit versuchen, diesen Overhead gering zu halten.

Bei der Projektabwicklung sind die zu Verfügung stehenden Ressourcen begrenzt. Die Arbeit eines Mitarbeiters kostet Zeit. Im Betrieb kostet sie Geld und der Mitarbeiter kann sich in dieser Zeit nicht anderen Tätigkeiten widmen, beim Studentenleben konkurriert die Arbeit an einem Projekt mit der Arbeit für andere Lehrveranstaltungen. Auch die Anzahl der Mitarbeiter ist beschränkt und gleichbleibend. Ebenso begrenzt sind die Arbeitsmittel und nicht zuletzt soll an einem gemeinsamen Produkt gearbeitet werden, das letztendlich nur ein mal vorkommt. Kurz zusammengefasst sollte die Projektorganisation also versuchen, diverse Problembereiche in den Griff zu bekommen:

- Zeitplanung und termingerechte Fertigstellung
- Aufgaben der Mitarbeiter definieren (z.B. verschiedene Programm-Module)
- den Tätigkeiten Ressourcen zuordnen (Arbeitsmittel wie Rechner, aber auch Programmcode)

In der Maschinenwelt wird diese Art der Projektorganisation vielleicht ausreichen. Wenn man es aber mit Menschen zu tun hat, ergeben sich weitere menschliche Schwierigkeiten:

menschliche  
Schwierigkeiten

- Einschätzung der eigenen und anderer Fähigkeiten
- Prestige einzelner Tätigkeiten
- (Un-)Lust
- Kommunikationsprobleme (»warme« Plauderei oder Höflichkeit vs. »kalte« Präzision)
- Einschätzung des Zeitbedarfs (z.B. zur Arbeit, aber auch die Zeit für die Fahrt zu einer Vorlesung oder einem Meeting)

Ab einer gewissen Größenordnung ist es ratsam, schon die Projektorganisation und die Qualitätskontrolle an einzelne Spezialisten zu übertragen, die im Ablauf Ressourcen verwalten, auf pünktliche Fertigstellung drängen oder zwischenmenschliche Probleme schlichten.

## 2 Prozessmodelle zur Software-Entwicklung

Software ist heutzutage so komplex, dass sie in den seltensten Fällen einfach programmiert werden kann. Es sind Vorüberlegungen anzustellen, damit die Software das leistet, was der Auftraggeber wünscht, und ihre Funktionsfähigkeit sicherzustellen.

### 2.1 Quick & Dirty

Das populärste Entwicklungsmodell unprofessioneller Programmierer ist der Quick & Dirty (Q & D) Ansatz: Die Software wird »mal eben schnell« programmiert, wenn man sich über den Zweck und den Weg der Programmierung im Klaren ist. Dazu wird in zwei Phasen vorgegangen:

1. Schreibe ein Programm
2. Finde und behebe die Fehler in deinem Programm

Sah sich ein Programmierer schon längere Zeit mit einem relativ kleinen Problem konfrontiert, dann hat er sich schon lange gedanklich damit befasst und es ist tatsächlich möglich, dass er nun mit dem Programmieren gute Software schreibt. Die Realität des Informatikers in seinem Berufsleben unterscheidet sich jedoch von dieser Situation, wenn er Software *für andere* schreiben soll oder die Funktion so umfangreich ist, dass die Software im Team entwickelt werden muss. Hier kennt er weder das Problem genau, das die Software lösen soll, noch hat er in ausreichendem Maße Lösungsansätze herausgearbeitet.

Spätestens bei der Erweiterung des Programms hätte man sich vielleicht etwas mehr konzeptionelle Weitsicht gewünscht, wenn entscheidende Teile neu geschrieben werden müssen oder man lästige Altlasten mit sich herumtragen muss.<sup>1</sup>

---

<sup>1</sup>Es gab einmal ein kleines Betriebssystem, das auf diese Weise geschrieben wurde: QDOS. Irgendwann wurde es von einem genialen Geschäftsmann aufgekauft, lediglich etwas verändert und einem IBM-PC beigelegt, womit es dann seinen Siegeszug rund um die Welt antrat. Und seitdem ärgern sich immernoch Benutzer mit unnötigen Altlasten herum... Welches Betriebssystem ist hier wohl gemeint?

## 2.2 Das Wasserfallmodell

Das oben genannte Modell wird nun ausgebaut: vor der eigentlichen Programmierung findet eine Planungsphase statt, in der der Rahmen des Projekts geplant wird. Dann wird im Rahmen der Definitionsphase der Ist-Zustand analysiert und die Ziele festgelegt. Während der Entwurfsphase werden die Ziele bis zu Konzepten heruntergebrochen und das neue System modellhaft entworfen, bis der Entwurf in den Code umgesetzt wird (Programmieren). Schließlich wird das Programm getestet und in Betrieb genommen.

Diese Phasen werden der Reihe nach durchgeführt. Es wird erst zur nächsten Phase gegangen, wenn eine Phase gänzlich abgeschlossen wurde. Stellt man einen Fehler fest, der in einer vorherigen Phase aufgetreten ist, so wiederholt man den Prozess von jener Phase an.

Dieses Modell bildet die Grundlage vieler anderer Vorgehensmodelle, da es logisch aufgebaut und durch seine Einfachheit klar und verständlich ist. Nachteil dieses Verfahrens ist, dass der Kunde lediglich an der Produktdefinition mitwirkt. Erst bei der Einführung erfährt er die Ergebnisse seines Auftrags und ist entweder zufrieden, oder eben nicht. Wenn nicht, kann es sein, dass der Entwicklung dermaßen daneben lag, dass gänzlich von vorne begonnen werden muss.

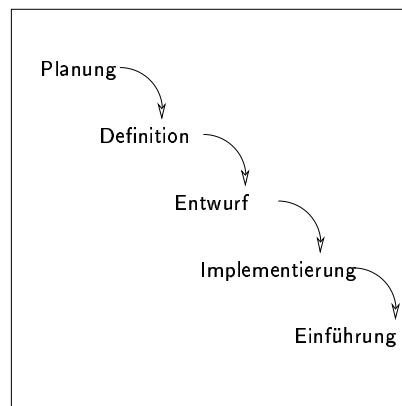


Abbildung 1: Das Wasserfallmodell

## 2.3 Das V-Modell

Das V-Modell erweitert das Wasserfallmodell dadurch, dass zahlreiche Verifikationen und Validierungen eingebaut werden: ausgehend von der fertigen Implementierung wird wieder zurückgegangen bis zu einem Abnahmetest beim Kunden.

## 2.4 Das Prototypen-Modell

Neben dem »normalen« (Wasserfall-) Projektablauf ist für das Prototypen-Modell charakteristisch, dass dem Kunden häufig Anschauungsmaterial präsentiert wird, aufgrund dessen er sich für den Auftragnehmer entscheiden, die bisherige Projektarbeit absegnen oder seine Wünsche zur Software präzisieren kann.

Welche Art Prototyp erstellt wird, hängt vom Zweck bzw. der gerade bearbeiteten Projektphase zusammen:<sup>2</sup>

**Demonstrationsprototyp:** Dieser Prototyp dient zur Auftragsakquisition und soll dem Auftraggeber lediglich einen Eindruck vermitteln, was ein Produkt leisten könnte. Solche Prototypen dienen allein der Vorführung und werden nach der Demonstration weggeworfen.

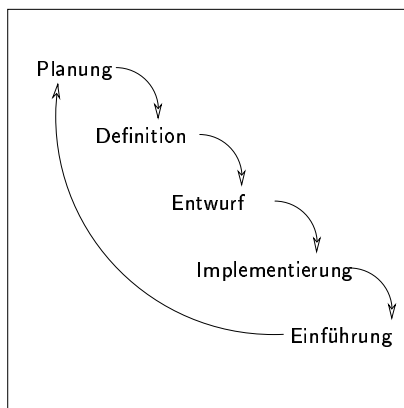
**Prototyp im engeren Sinne** ist ein Prototyp, an dem spezielle Aspekte praktisch getestet werden sollen.

<sup>2</sup>Vgl. [1] S. 115

**Labormuster** sind Prototypen, die den Entwicklern ein Beispiel für eine bestimmte Funktionalität geben sollen. Im Rahmen des Praktikums dieser Lehrveranstaltung werdet ihr (hoffentlich) viele »Labormuster« erhalten, um ein Anschauungsmodell über bestimmte Funktionalitäten zu bekommen.

**Pilotsystem:** Dies ist ein Prototyp, der nicht nur der experimentellen Beobachtung dient, sondern auch ein Teil bzw. der Kern des Endprodukts ist.

## 2.5 Das evolutionäre Modell



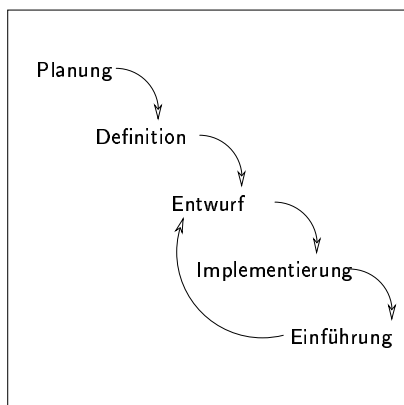
**Abbildung 2:** Das evolutionäre Modell

Damit lässt sich vermeiden, dass der Auftragnehmer komplett am Kunden vorbeientwickelt und auf seinem Produkt sitzenbleibt. Auch im Bereich des »anonymen« Kunden (Standardsoftware) wird häufig auf diese Weise entwickelt.

Das evolutionäre Modell sieht vor, dass es sich beim ersten Produkt noch nicht um das eigentliche Endprodukt handeln muss. Das Ziel ist, so früh wie möglich den Kern des Programms zu implementieren und dann fehlende Teile in erneuten Produktionszyklen zu entwickeln. Damit können die Erfahrungen und Wünsche des Kunden und der Benutzer in spätere Programmversionen einfließen.

Durch den kleineren Rahmen ist es sehr viel schneller möglich, dem Kunden etwas auszuliefern. Außerdem kann man durch Rückkopplung von den Anwendern erfahren, in welche Richtung das Produkt weiterentwickelt werden soll. Da-

## 2.6 Das inkrementelle Modell



**Abbildung 3:** Das inkrementelle Modell

Während das evolutionäre Modell auch den Analysebereich auf das Gebiet einschränkt, das implementiert wird, analysiert das inkrementelle Modell den Problembereich vollständig, entwickelt dann aber das Produkt in Aufbaustufen.

So soll gewährleistet sein, dass man vorhandene Teile des Programms nicht etwa neu schreiben muss, wenn man bei einem erneuten Durchlauf der Definitionsphase andere Modelle wählt. Durch den ausführlicheren Anfang des Projekts kommt man aber auch nicht so schnell zu einem ersten Prototyp.

## 2.7 Das objektorientierte Modell

Die bisherigen Prozessmodelle verwendeten alle einen »top down«-Ansatz, d.h. sie entwarfen das Programm nach den Zielen, nicht nach den Möglichkeiten. Charakteristisch für das objektorientierte Modell ist die Wiederverwendung bereits bestehender Programmteile (Klassen, Objekte). Daher wird hier eine »bottom up«-Komponente ins Modell aufgenommen. Das Design der Software richtet sich also auch nach bereits vorhandenen Programmteilen. Das objektorientierte Modell wird häufig in Verbindung mit evolutionären Ansätzen kombiniert.

## 2.8 Das nebenläufige Modell

Das nebenläufige Modell sieht vor, dass ein Programm stufenweise aufgebaut wird, die einzelnen Stufen aber parallel in Teams aufgebaut werden.

## 2.9 Das Spiralmodell

Das Spiralmodell gibt nur einen Rahmen vor, welche Fragen zu stellen sind, um ein geeignetes Prozessmodell zu entwerfen, zu hinterfragen und abzuändern. Da es somit einen hohen Managementaufwand bedeutet, eignet es sich nicht für kleine bis mittlere Anwendungen und Teams und soll hier nicht weiter ausgeführt werden.<sup>3</sup>

## 2.10 Agile Entwicklungsmethoden

*(Dr. Peter Rüßmann)*

## 2.11 Extreme Programming (XP)

*(Irina Gimpeliovskaja und Susanne Richter)*

## 2.12 Open-Source-Softwareentwicklung

*(Jan-Felix Breuer und Marco Rademacher)*

Open-Source-Softwareentwicklung im weitesten Sinne ist ein Entwicklungsprozess, bei dem der Quellcode der Software der Öffentlichkeit zugänglich ist.<sup>4</sup> Im engeren Sinne bedeutet es einen Entwicklungsprozess, bei dem sowohl die Programmquellen, als auch die Projekt-relevanten Entscheidungsprozesse öffentlich zugänglich sind: »Jeder interessierte Anwender hat die Möglichkeit den weiteren Projektverlauf aktiv zu beeinflussen.«

Da sich ein Open-Source-Projekt über ein besonderes Verhältnis zu seinen Anwendern definiert, kann also erst von einem Open-Source-Entwicklungsprozess gesprochen werden, wenn sich eine interessierte und involvierte Anwender-Gemeinde gebildet hat. Bei fast allen Open-Source-Projekten geschieht dies erst nach der Veröffentlichung eines brauchbaren Prototyps. Open-Source-Entwicklung bezieht sich also in fast allen Fällen auf die evolutionäre Weiterentwicklung einer bestehenden (unter Umständen rudimentären) Software und nicht auf die Erstellung eines Prototyps. Auch kennt Open-Source-Entwicklung kein »fertiges« Produkt: Wird ein Projekt vom Maintainer nicht

---

<sup>3</sup>Mehr in [1].

<sup>4</sup>Der öffentliche Zugang wird technisch meist über eine Website gelöst und juristisch über eine Open-Source-Lizenz, wie z.B. der GNU General Public License (GPL), abgesichert.

mehr aktiv entwickelt, ist dies Anlass für die Anwender, sich des Projekts entweder selbst anzunehmen oder ein anderes, vergleichbares zu verwenden.

Meist werden Open-Source-Projekte von Privatleuten betrieben, die über das Internet miteinander kooperieren. Die Motivation erwächst nicht aus finanziellem Interesse, sondern aus dem Bedürfnis ein Problem zu lösen »das einen persönlich juckt«. <sup>5</sup> Die Motivation für die Arbeit an einer Problemlösung kann wohl kaum größer sein.

Motivation

Wird ein Projekt erst einmal von einer Nutzer-Gemeinde angenommen, entsteht häufig eine positive Rückkopplung: Anwender machen dem Maintainer bewusst, dass seine Arbeit geschätzt wird. Oft tragen sie außerdem aus Eigeninteresse etwas bei (Ideen, Bug-Reports/-Fixes, Dokumentation, Erweiterungen, Entwicklungs-Arbeit, etc.), da sie so das Projekt in ihrem Sinne verbessern. Solche Beiträge entlasten den Maintainer von Programmierarbeit und beschleunigen die Entwicklung. Es liegt also im Interesse des Maintainers, Beiträge seiner Anwender anzunehmen und ihnen sogar Aufgaben zu übertragen: In einem Open-Source-Projekt ist der Übergang von Anwender zu Entwickler fließend. Ein erfolgreiches Projekt mit steigender Anwender-/Entwicklerzahl kann so zum Selbstläufer werden und der Maintainer übernimmt zunehmend Koordinierungs- und Entscheidungsfunktion.

Die Planung eines Projekts (sowohl in Detailfragen, wie auch im großen und ganzen) findet öffentlich (meist über eine Mailingliste) statt. Zentral ist, dass dabei Anwender, Entwickler und der Maintainer gleichberechtigt diskutieren. So hat jeder die Möglichkeit neue Ideen vorzubringen und es wird jede Idee, auch Ideen des Maintainers, gründlich diskutiert ("Peer Review"). Diese offene Diskussionsform führt bemerkenswerterweise auch bei vielen Beteiligten nur zu unwesentlichen Reibungsverlusten. Bleiben unterschiedliche Positionen auch nach der Diskussion bestehen, so hat der Maintainer das letzte Wort. Wenn einem Nutzer jedoch genug an seiner abweichenden Idee liegt, so räumen ihm die meisten Open-Source Lizenzen das Recht ein, einen Ableger ("Fork") des Projekts nach den eigenen Vorstellungen weiterzuführen.

Entscheidungsprozess

Zusammenfassend läuft eine erfolgreiche Open-Source-Softwareentwicklung folgendermaßen ab:

Ablauf

- Jemand fängt an, Software zu schreiben, die ein eigenes Problem lösen soll.
- Die aktuelle Version des (funktionierende) Prototyp wird veröffentlicht. Andere Anwender mit demselben Problem werden auf die Software aufmerksam.
- Anwender beteiligen sich aus Eigeninteresse und werden zu Mitentwicklern: »Behandle deine Anwender so, als wären sie deine wertvollste Ressource, und sie werden deine wertvollste Ressource.« <sup>6</sup>
- Die Software kann bei Ablehnung von Beiträgen oder einschlafendem Interesse des Maintainers anderen komplett übergeben oder von ihnen als eigener Zweig weitergeführt werden.

Mehr zu diesem Thema in [3] und in einer späteren Vorlesung zu Open-Source und CVS.

<sup>5</sup>Eric Raymond ebd: »Every good work of software starts by scratching a developer's personal itch.«

<sup>6</sup>Eric Raymond in [4] »If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.«

## 2.13 EASE

EASE<sup>7</sup> (Education for Actual Software Engineering) ist ein von Dirk Draheim vorgeschlagener Entwicklungsprozess speziell für universitäre Lehre. Gerade an der Uni sieht man sich oft dem Problem konfrontiert, dass Projekte nicht zu Ende geführt werden und daher für alle Beteiligten eher unbefriedigend sind. Verantwortlich dafür ist einerseits die Fixierung auf das Produkt, andererseits auch der gute Wille der Teilnehmer zum Beginn des Projekts und den während des Semesters zunehmenden Zeitproblemen, da auch für andere Lehrveranstaltungen gearbeitet werden muss.

Diese Probleme versucht EASE durch den Ansatz zu beheben, indem der Fokus des Uni-Projekts auf das Lernen gelegt wird und die Fertigstellung des eigentlichen Produkts zur Nebensache wird. Folgende Merkmale charakterisieren den EASE-Prozess:

- Projekt-Ziel ist das Lernen, nicht das Produkt
- Jeder Student arbeitet im Umfang eines Zeitkontingents an den Aufgaben, unabhängig vom Fortschritt des Produkts
- Insofern gibt es keine Meilensteine, die sich am Fortschritt des Produkts orientieren, sondern »Checkpoints«
- Durch Rotation der Teilnehmer(-Gruppen) über die Aufgabenbereiche in bestimmten Intervallen ist jeder Teilnehmer auch wirklich mit jedem Thema in Kontakt gekommen

Insbesondere die Rotation verlangt von den Teilnehmern, die Nachfolger in das eigene Gebiet einzuweisen. Gefördert wird damit auch die Herstellung von Dokumentation (je besser der Hilfetext, desto geringer ist die Nachfrage nach Hilfestellung), aber auch das Lernen an sich, denn gerade beim Erklären als aktivem Prozess wird mit am meisten gelernt.

Nachteil der Reinform von EASE ist natürlich, dass gerade der Einblick in typische Probleme üblicher Entwicklungsprojekte ausgeblendet werden.<sup>8</sup>

## 2.14 Abschließende Betrachtung

Welches ist nun das richtige Modell? Wie bei allen Modellen gibt es keine Wahrheit, sondern nur Zweckmäßigkeit. Welches Modell gewählt werden soll, hängt von hohem Maße von der Eigenart des Projekts, dem Auftraggeber und auch dem Entwickler selber (und seiner Flexibilität) ab. Es gibt einfache Modelle, die nicht viel mit der Realität zu tun haben, aber sehr verständlich sind, sodass man gut damit arbeiten kann. Und es gibt komplexe Modelle, die besser auf die Realität guter Softwareentwicklung passen, aber so schwer verständlich sind, dass man kaum mehr damit arbeiten kann. Hier ist also von jedem Entwicklungsteam und dem Kunden eine Entscheidung zu treffen, wie vorgegangen werden soll, und das Prozessmodell dem jeweiligen Projekt anzupassen.

---

<sup>7</sup><http://www.inf.fu-berlin.de/projects/ease/>

<sup>8</sup>Mehr zu EASE in [2].

## 3 Zusammenarbeit mit dem Kunden

»Auftraggeber kommen vom Mars, Auftragnehmer von der Venus«<sup>9</sup>

Während sich der Dienstleister gerne in sein Spezialgebiet vertieft (schnelle Programme, originelles Design, spektakuläre Ideen, usw.) hat der Kunde wahrscheinlich ein gänzlich anderes Interesse: die Software ist für ihn nicht einfach Spielerei, sondern kann über den Erhalt einer Firma entscheiden, indem mit der Software der Umsatz gesteigert werden soll. Auch in unserem Anwendungsgebiet, also im Schulbereich, hat der Kunde ein professionelles Interesse. Er will Software für einen bestimmten Einsatzzweck, nicht für Spielereien und Beschäftigung einiger Programmierer. Er erwartet, dass der Dienstleister sein Problem löst. Nicht mehr und nicht weniger.

### 3.1 Das Selbstverständnis der Dienstleistung

Für den Dienstleister heißt das, sich selbst möglichst zurückzuhalten. Gute, innovative Ideen sind natürlich mehr als erlaubt, insbesondere wenn der Kunde noch nicht genau weiß, was er will. Hier kann der Sachverstand der Informatiker auf der Seite der Software ein Anlass sein, ihre Ideen und Kenntnisse einzubringen, um ein sinnvolles Produkt zu schaffen. Man sollte sich aber im Klaren darüber sein, dass niemand so leicht über seinen eigenen Schatten springen kann und man sich wahrscheinlich doch mit seiner Lieblingsbeschäftigung oder problembehafteten Arbeitsschritten verzettelt. Daher ist eine möglichst externe Qualitätskontrolle, also eine Person, ein Gremium oder ähnliches, das in Regelmäßigkeit über den Ablauf des Projekts wacht, eine sinnvolle Einrichtung, um nahe an den Zielen und dem Zeitrahmen des Kunden zu bleiben und sich nicht zu verzetteln.

### 3.2 Auftraggeber und Anwender

Der Kunde besteht meist aus einem Entscheidungsträger der Führungsebene einer Firma. Ab einem bestimmten Punkt ist er bestrebt, die eigentliche Arbeit zu delegieren. Sodann wird mit dem Mittelbau kommuniziert, der sich oft dadurch auszeichnet, mehr Sachkenntnis zu haben, aber über weniger Entscheidungskompetenz zu verfügen. Ferner sind Mitarbeiter aus dem Mittelbau scheinbar noch nicht am Ende ihrer Karriere: bei ihrer persönlichen »Politik« spielen deshalb auch persönliche Karrierepläne eine Rolle. Ob sich der Dienstleister darauf einlässt, diese Eitelkeiten zu befriedigen oder nicht, hängt sicherlich von jedem einzelnen ab. Entscheidend ist aber, das persönliche Wunschenken der Einzelnen zu erkennen und von den Bedürfnissen des Auftraggebers und seinen Zielen zu trennen.

Oft wird bei der Systementwicklung aber die entscheidende Ebene unterschlagen, nämlich die Anwender. Der normale Angestellte arbeitet nicht so sehr für das Firmenwohl, wie für das eigene Auskommen. Ihn interessiert die Umsatzsteigerung nicht so sehr wie der Erhalt seines Arbeitsplatzes. Solange der Computereinsatz nicht nur mit Steigerung der Produktivität, sondern auch mit Entlassung einhergeht, muss sich der Systemanalytiker nicht wundern, wenn er bei den Anwendern auf Widerstände stößt. Solche Widerstände können einfach nur ärgerlich oder zeitraubend sein – sie können aber auch das Ergebnis der Analyse dahingehend verfälschen, dass die Entwicklung der Software auf gänzlich falschen Annahmen beruht. Insofern muss das Bestreben des Informatikers dahin gehen, Vertrauen zu schaffen, um Wahrheit zu erfahren.

---

<sup>9</sup>Aus: [5]

### 3.3 Wahrheit, Vertrauen, Transparenz

Um das richtige Softwareprodukt zu entwerfen, müssen die Voraussetzungen richtig sein. Gemeinhin ist die Analyse bestehender Systeme eine Grundvoraussetzung vor dem Design eines neuen Systems. Hier ist der Systemanalytiker also auf wahre Angaben der Anwender angewiesen. Um nicht als Jobkiller oder einfach unliebsamer Besucher zu erscheinen, ist es wichtig, die eigenen Absichten so zu formulieren, dass die Anwender Vertrauen entwickeln können. Auch der Kunde (die Chefetage) ist daran interessiert, auf eine ordnungsgemäße Arbeit der Informatiker vertrauen zu können. Dieses Vertrauen kann nur mit Kommunikation und ständiger Präsenz aufgebaut und gehalten werden. Was sich hier anbietet, ist (mal wieder) das Internet.

Auf einer Website kann der Dienstleister seinem Kunden darstellen, wie er das Projekt geplant hat und wie es tatsächlich bisher ablief. Mit solcher Dokumentation hilft er nicht nur seinem Kunden, den Blutdruck zu senken, er kann sich auch damit selbst entlasten, da der Kunde nicht so oft »sicherheitshalber« nachfragen muss. Außerdem zwingt solche – natürlich ehrliche – Ablaufdokumentation den Informatiker, das Projekt gut zu planen und erfährt schneller, wenn etwas nicht so läuft, wie er es plante, und kann schon früh geeignete Maßnahmen ergreifen.

In großen Projekten oder großen Firmen wird solche Qualitätskontrolle auch oft ausgelagert, um hier verschiedene Aufgaben auch verschiedenen Rollen zuzuordnen, die sich dann voreinander rechtfertigen müssen.

## Aufgaben zum Projektfortgang

Was könnte für einen erfolgreichen Projektverlauf geplant werden?

### Literatur

- [1] Helmut Balzert  
**Lehrbuch der Software-Technik**  
Spektrum Akademischer Verlag, Heidelberg, 1998
- [2] Dirk Draheim  
**Learning Software Engineering with EASE**  
Kluwer Academic Publishers  
  
Ein Exemplar dieser Arbeit könnt Ihr bei mir oder direkt bei Dirk erhalten.
- [3] Karl Fogel  
**Open-Source-Projekte mit CVS**  
MITP-Verlag, Bonn, 2000
- [4] Eric Raymond  
**The Cathedral and the Bazaar**  
<http://www.catb.org/~esr/writings/cathedral-bazaar/>
- [5] David Siegel  
**Das Geheimnis erfolgreicher Websites – Business, Budget, Manpower, Lizenzen, Design**  
Markt und Technik Verlag, München, 1998