Klassische Fehler in der Software-Entwicklung

CEFE - CAD/CAM-Entwicklungsgesellschaft
Arbeitsgruppe 19 Software Engineering*

1 Einleitung

In diesem Artikel werden einige typische Fehler behandelt, die im Rahmen der Software-Entwicklung häufig auftreten. Anhand von allgemein gehaltenen Beispielen zu den Ursachen, den möglichen Folgen und entsprechenden Maßnahmen soll das Problembewußtsein der Entwickler und ihrer Vorgesetzten zu den einzelnen Punkten gestärkt werden. Obwohl viele dieser Fehler zwar seit langem bekannt sind, ist die Vermeidung dieser Fehler trotzdem weitaus schwieriger, als es auf den ersten Blick scheint. Bei immer kürzer werdenden Entwicklungszyklen und ständig steigender Komplexität der äußeren Rahmenbedingungen muß mit dem zur Verfügung stehenden Budget in möglichst kurzer Zeit qualitativ hochwertige Software entwickelt werden, die außerdem noch entsprechenden Konkurrenzprodukten überlegen ist. Die Kosten für die Beseitigung von Produktmängeln stellen in der Regel direkt entgangenen Gewinn dar. Der Entwicklungsprozeß ist deshalb unter vertretbarem Aufwand so zu gestalten, daß Mängel gar nicht erst auftreten.

2 Übersicht

Die einzelnen Fehler werden zunächst in Kategorien eingeteilt, die zum Teil mit den Phasen des Software-Entwicklungssprozesses identisch sind. Anschließend werden im nächsten Kapitel zu jedem der Fehler verschiedene Ursachen, Folgen und Maßnahmen zur Vermeidung dieser Fehler im Detail ausgeführt.

Vorgehensmodell:

- Es wird direkt mit der Codierung angefangen.
- Das Vorgehensmodell fehlt bzw. wird nicht befolgt.
- Die Terminvorgaben sind unrealistisch.
- Die Weiterbildung der Mitarbeiter ist nicht zielgerichtet.
- Auswahl und Einsatz der Werkzeuge bzw. Methoden ist unzureichend vorbereitet.
- Ein Risikomanagement wird nicht betrieben.

Verifizierung und Validierung:

- Eine Abnahme der Phasenergebnisse erfolgt nicht.
- Es wird nicht systematisch bzw. unzureichend getestet.

Problem- und Systemanalyse:

- Die Anforderungen und Qualitätsmerkmale werden nicht festgelegt.
- Es fehlen eindeutige Begriffsdefinitionen.

Systementwurf:

- Die Systemarchitektur ist gar nicht oder nur mit großem Aufwand erweiterbar.
- Das System ist nicht modular aufgebaut, die Daten sind nicht gekapselt.

Codierung:

- Programmier-Standards bzw. -Richtlinien werden nicht beachtet.
- Die Namensvergabe ist ungünstig.

Betrieb und Wartung:

- Die Dokumentation fehlt ganz, ist veraltet oder nicht adäquat.
- Die Schulung der Anwender wird vernachlässigt.
- Das Konfigurationsmanagement ist unzureichend.

3 Erläuterungen

3.1 Vorgehensmodell

Das Vorgehensmodell zerlegt den Software-Entwicklungsprozeß in überschaubare Phasen und definiert Aktionen und Ergebnisse. Erst dadurch kann der Schritt von "chaotischer Software-Bastelei" zur industriellen Software-Produktion vollzogen werden. Ist ein unstrukturiertes Vorgehen bei Quick & Dirty Wegwerf-Software noch zu tolerieren, so hat das bei allen Systemen, die einer Wartung unterliegen, vernichtende Konsequenzen. Das Fehlen oder

*Specher der Arbeitsgruppe: Dierk Wendt, GDV-E, Hella KG, 59552 Lippstadt, 02941/387614, wendt@hella.de

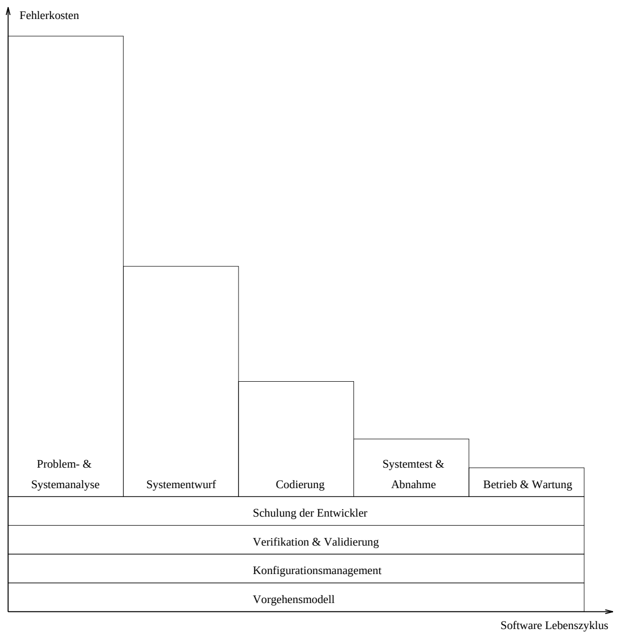

Abbildung 1: Einflußfaktoren auf die Fehlerkosten betrachtet über den Lebenszyklus

auch Nichtbefolgen eines Vorgehensmodells bedeutet, daß der Software-Entwicklungsprozeß nicht beherrscht wird.

Ein Vergleich: Ein Schreiner, der eine Tür bauen soll, nimmt das Maß auf, läßt sich die Details der Tür vom Bauherren (schriftlich) bestätigen, fertigt eine Skizze an, stellt Tür und Zarge in der Werkstatt her, geht zur Baustelle und setzt sie ein. Käme der Mann aus der Software-Entwicklung, unterläge er der Versuchung, sein Werkzeug und einen großen Stapel Holz aus dem Lager an die Baustelle zu transportieren und dort solange zu sägen, bohren, schrauben und hämmern, bis das Loch in der Wand geschlossen ist, mit mehr oder minder zufälligem Ergebnis.

Und dort liegt die Krux in der Software-Entwicklung. Die Umgebung gestattet es in der Regel, bis zum letzten Moment an dem Produkt zu ändern, ohne daß man es diesem sofort ansieht, während der Schreiner aus dem Beispiel ohne sein strukturiertes Vorgehen den Auftrag nicht mit wirtschaftlichem Erfolg und für den Kunden zufriedenstellend abschließen könnte. Das geht auch in der Software-Entwicklung nicht, nur merkt man es nicht so schnell!

Software-Projekte laufen immer Gefahr, Termin und Budget zu überschreiten. Durch die Zerlegung in überschaubare Phasen gewinnt man mit einem Vorgehensmodell Planungssicherheit und erhält durch die eingebauten Meilensteine und Verifizierungen ein Lenkungsinstrument, mit dem frühzeitig Abweichungen erkannt und Korrekturmaßnahmen ergriffen werden können. Eine Koordination mehrerer in das Projekt eingebundener Personen oder Stellen ist ohne Vorgehensmodell praktisch nicht möglich. So wird oft bei großen und kritischen Projekten das anzuwendende Modell vom Auftraggeber (z.B. ESA) vorgeschrieben.

3.1.1 Es wird direkt mit der Codierung angefangen

- Ursachen:
  - Dies ist wohl der häufigste Fehler in der Softwareentwicklung. Dieses Vorgehen bringt dem Entwickler vordergründig einen Zeitgewinn. Er kann sehr schnell etwas vorweisen und eine Lösung präsentieren.
- Folgen:
  - Das Umsetzen der ersten Ideen direkt in Source Code ist immer unzulänglich, da der wichtigste Schritt – das Systemdesign – umgangen wird. Selbst für eine prototypische Umsetzung ist das Vorgehen nicht geeignet, da ohne Betrachtung der Systemumgebung und möglicher Realisierungsalternativen auch die Aussagekraft des Prototyps sehr zweifelhaft ist. Ohne Dokumentation des gedanklichen Weges, der zu dem Software-Produkt geführt hat, ist dieses für Außenstehende und nach einiger Zeit oft auch für den Autor nicht mehr nachvollziehbar. Selbst mit entsprechender Dokumentation ist dies oft schwierig genug. Sogenannte Prototypen bergen zudem immer das Risiko, plötzlich aus Zeitdruck zu einem Produkt befördert zu werden.
- Maßnahmen:
  - Die direkte Codierung muß rigoros auf Wegwerf-Software begrenzt werden (z.B. kurz benötigte Testumgebung, "workarounds").
  - Seitens der Führung muß sichergestellt werden, daß strukturiert vorgegangen wird. Dazu muß ein Vorgehensmodell oder -plan existieren, der die Entwurfsphasen berücksichtigt.
  - Der Entwickler muß angehalten werden, Entwurfsdokumente zu erstellen. Diese müssen zeitlich geplant, verabschiedet und aktuell gehalten werden.

3.1.2 Das Vorgehensmodell fehlt bzw. wird nicht befolgt

- Ursachen:
  - Die Notwendigkeit wird nicht erkannt: "Hier weiß jeder, was zu tun ist". Das kann für eine geschlossene Gruppe mit einer starken Führung, die die Fäden in der Hand hält, zutreffend sein. In diesem Fall wird implizit nach einem ungeschriebenen Vorgehensmodell entwickelt.
  - Die Leitung nutzt ihrerseits nicht die Steuerinstrumente, die das Vorgehensmodell bietet.

Es ist ein Papiertiger und wird deshalb nicht befolgt.
  - Das Arbeiten nach einem Vorgehensmodell macht die Arbeit transparent. Das mögen viele Entwickler aus unterschiedlichen Gründen nicht.
- Folgen:
  - Die Planung ist unsicher.
  - Die Leitung verzichtet auf ein wirksames Instrumentarium, um Projekte erfolgreich zu lenken. Probleme werden deshalb zu spät erkannt.
  - Die Gefahr von Qualitätsproblemen steigt.
  - Teamwork und die Kommunikation zwischen den Teams wird erschwert.
  - Die Einarbeitung in ein fremdes Projekt dauert länger, auch für neue Mitarbeiter.
- Maßnahmen:
  - Falls noch nicht existent, muß ein Vorgehensmodell definiert und eingeführt werden. Dabei spielt es keine Rolle, welches Vorgehensmodell (Wasserfall, V-Modell, Spiralmodell, ...) als Basis ausgewählt wird. Wichtig ist dabei, daß alle Entwickler ihre Tätigkeit darin wiederfinden müssen und das Modell auf alle Projekte je nach Größe und Risiko konfektionierbar ist, zum Beispiel durch die Unterscheidung zwischen obligatorischen und fakultativen Meilensteinen.
  - Der positive Effekt des Vorgehensmodells auf die Qualität ist allen Beteiligten deutlich zu machen.

3.1.3 Die Terminvorgaben sind unrealistisch

- Ursachen:
  - Es wird nach dem Motto verfahren, das Unmögliche fordern, um das Mögliche zu erreichen.
  - Die Einbeziehung der DV-Aspekte in die Projektplanung ist mangelhaft.
  - Die Qualifikation in Planungs-, Organisations- und Führungstechniken ist zu gering.
  - Der meist große Zeitbedarf für Verifizierung und Validierung wird weit unterschätzt.
- Folgen:
  - Es herrscht Frust und Pessimismus bei den beteiligten Mitarbeitern.
  - Es gibt Akzeptanzprobleme bezüglich der Zielvorgaben, Maßnahmen und Leitungsstrukturen.
  - Die Mitarbeiter fühlen sich übergangen.
- Maßnahmen:
  - Es sind realistische Ziele zu vereinbaren.
  - Alle betroffenen Stellen im eigenen Hause sowie ggfs. weitere beteiligte Unternehmen sind frühzeitig in die Terminplanung einzubeziehen.
  - Qualifizierung der entsprechenden Mitarbeiter in formellen Planungs-, Organisations-, Controlling- und Führungstechniken.
  - Gewissenhafte und umfassende Information aller Mitarbeiter, soweit vertretbar. Fehlende offizielle Information fördert die Entstehung und Fehlinterpretation inoffizieller Informationen.

3.1.4 Die Weiterbildung der Mitarbeiter ist nicht zielgerichtet

- Ursachen:
  - Weiterbildung betrifft oft zukünftige Aufgaben, die häufig aufgrund kurzfristiger Überlegungen dem aktuellen Tagesgeschäft untergeordnet wird.
  - Die Mitarbeiter werden nicht als strategische Ressource angesehen.
  - Kosten und Termingründe.
  - Weiterbildungsmaßnahmen, insbesondere extern durchgeführte, werden zum Teil als Prestigesymbol angesehen. Firmenspezifische Belange sind dabei nebensächlich.
- Folgen:
  - Es gibt keine systematische Fortbildung, insbesondere fehlt Training-on-the-job. Ohne geplante Weiterbildungsmaßnahmen entspricht die Qualifikation der Mitarbeiter langfristig nicht mehr den Anforderungen.
  - Die Mitarbeiter greifen zum Selbststudium. Dies ist in der Regel mit großem Zeitverlust verbunden. Außerdem fehlt oft der thematische Gesamtüberblick aufgrund von lückenhaftem Wissen.
- Maßnahmen:
  - Eine gründliche Ausbildung in den verwendeten Methoden, Verfahren und Werkzeugen ist eine wichtige Voraussetzung für positive Arbeitsergebnisse. Deshalb ist es angebracht, die Entwicklung des Produktionsfaktors "Arbeit" ebenso systematisch zu managen, wie die anderer Produktionsfaktoren.
  - Externe Schulungen können firmenspezifischen Weiterbildungsbedarf nicht immer befriedigend decken. Es kann daher sinnvoll sein, durch befristete, projektbezogene Einbeziehung externen Expertenwissens (z.B. Hochschulen, Unternehmensberatungen usw.) ein Training-on-the-job zu realisieren. Wichtig hierbei ist, daß ein Know-how-Transfer sichergestellt wird.

- Ein Erfahrungsaustausch ist oft im Rahmen von firmenübergreifenden Workshops oder Arbeitskreisen möglich. Diese werden von Berufsorganisationen, Universitäten usw. angeboten.

3.1.5 Auswahl und Einsatz der Werkzeuge bzw. Methoden ist unzureichend vorbereitet

- Ursachen:
  - Bei Projektbeginn sind die Anforderungen an die Werkzeuge bzw. Methoden noch unklar.
  - Moderne Software-Entwicklungswerkzeuge sind z.T. sehr komplex; ihre Beurteilung und Auswahl ist daher zeitaufwendig.
  - Der Weiterbildungsbedarf der Mitarbeiter bezüglich neuer Technologien wird unterschätzt.
- Folgen:
  - Ungeeignete Methoden und Werkzeuge führen zu Produktivitäts- und Qualitätseinbußen.
  - Die Mitarbeiter versuchen, inadäquate Werkzeuge zu umgehen.
  - Die Fehlinvestionen für falsche Werkzeuge müssen oft teuer bezahlt werden (Wartungskosten alter Daten, Anschaffungs- und Schulungskosten für das neue Werkzeug).
- Maßnahmen:
  - Neue Unternehmensstrategien können eine Anpassung von Methoden und Werkzeugen notwendig machen. Dies ist bei der Planung und Auswahl zu berücksichtigen.
  - Die Einbeziehung von externem Expertenwissen ist oft sinnvoll. Das entscheidende Kriterium ist dabei die nachgewiesene Praxiserfahrung des Beraters und nicht ein wohlklingender akademischer Titel.
  - Kriterienkataloge für die Auswahl von Werkzeugen bzw. Evaulationsberichte von Unternehmensberatungen können wichtige Hinweise liefern. Weitergehende Informationen bieten Workshops oder Anwenderkonferenzen.
  - In einem Pilotprojekt sind ein oder zwei der in Frage kommenden Werkzeuge bzgl. ihrer Leistungen im "Alltagsgeschäft" gründlich zu testen.

3.1.6 Ein Risikomanagement wird nicht betrieben

- Ursachen:
  - Der Kosten- und Termindruck ist zu groß.
  - Problempotentiale, insbesondere fachfremde Aspekte wie die Produkthaftung, werden falsch eingeschätzt.
  - Risiken werden "aktiv verdrängt".
  - Es fehlt eine Datenbasis mit brauchbaren Erfahrungswerten.
- Folgen:
  - Planungen sind unzuverlässig, da sie Risikopotentiale nicht berücksichtigen. Durch unvorhergesehene Probleme werden Terminverzögerungen und Kostenüberschreitungen verursacht.
  - Es fehlen Regelkreise im Vorgehensmodell, um die Phasenergebnisse zu validieren. Die konstruierten Systeme gehen u.U. an den Realwelt-Gegebenheiten (Datenumfang, Schnittstellen, Kundenwünsche usw.) vorbei.
- Maßnahmen:
  - Risikomanagement ist erlernbar. Projekte scheitern nicht an den Dingen, die beherrscht werden oder schon ähnlich konstruiert wurden. Problematisch sind in der Regel nur neue Technologien. Deshalb ist die Integration des Risikomanagementes in ein Vorgehensmodell (z.B. das Spiralmodell nach Boehm) wichtig.
  - Benchmarking und Prototyping sind wichtige Methoden zum identifizieren und abschätzen von Risiken.
  - Fremdleistungen (z.B. Software-Module) sollten ständig überwacht werden. Hier darf die Qualitätssicherung nicht vernachlässigt werden.
  - Wenn kein ausreichendes eigenes Know-how vorhanden ist, kann möglicherweise externe Erfahrung genutzt werden.

3.2 Verifizierung und Validierung

Qualitätsprüfungen müssen in jeder Phase des Entwicklungsprozesses durchgeführt werden. Idealerweise sollte jedes Zwischenprodukt überprüft werden: Definition, Spezifikation, Entwurf, Code, Benutzer-Dokumentation, mit einem Wort: alles.

Um diese Prüfungen und Tests durchführen zu können, muß selbstverständlich vorher definiert werden, welche (Qualitäts-)Eigenschaften die jeweiligen Zwischenprodukte haben sollen. Das heißt, daß aus den Anforderungen an das Endprodukt die Anforderungen an das jeweilige Zwischenprodukt abgeleitet werden müssen.

Qualitätssicherung wird in der Software-Entwicklung durch Verfikations- und Validierungs-Aktivitäten geleistet. Bei der Verifizierung wird die Frage "Haben wir das System richtig entwickelt?" beantwortet, d.h. es wird geprüft, ob es die Anforderungen der Spezifikation der jeweiligen Entwicklungsphase erfüllt. Die Validierung hingegen soll die Frage "Haben wir das richtige System entwickelt?" beantworten; es wird überprüft, ob die Anforderungen des Kunden erfüllt werden.

Wir können und sollten prüfen

- jede Spezifikation bezüglich ihrer internen Folgerichtigkeit und Widerspruchsfreiheit,
- jede Spezifikation (außer der ersten) in Hinblick auf die Vorgänger- Spezifikation (Vollständigkeit, genaue funktionale Entsprechung),
- jedes Programm-Modul bezüglich der zugehörigen Spezifikation auf Vollständigkeit und funktionale Gleichwertigkeit,
- jedes Programm-Modul hinsichtlich der internen Folgerichtigkeit und Widerspruchsfreiheit

3.2.1 Eine Abnahme der Phasenergebnisse erfolgt nicht

- Ursachen:
  - Die Anforderungen und Qualitätsmerkmale wurden nicht festgelegt.
  - Ein geeignetes Vorgehensmodell fehlt.
  - Durch häufige Iterationen im Entwicklungsablauf ist ein Phasenabschluß nicht immer eindeutig erkennbar.
- Folgen:
  - Fehler werden erst in späteren Phasen oder sogar erst beim Kunden entdeckt.
  - Die Kosten steigen infolge verzögerter Fehlerentdeckung exponentiell an.
  - Der Projekt-Fortschritt kann nicht beurteilt werden, da über die Qualität der vorliegenden Produkte keine durch Prüfungen bestätigte Aussagen vorliegen.
- Maßnahmen:
  - Die Zeitpunkte und die Voraussetzungen für die Abnahme der Phasenergebnisse müssen im Vorgehensmodell bzw. dem Projekt-Plan eindeutig festgelegt werden.
  - Sehr gut geeignet für die Überprüfung der Phasenergebnisse sind Review-Verfahren. Die Vorteile des Review-Verfahrens zeigt die nachstehende Tabelle [7]:

Entwicklungsphase	Gefundene Fehler
Review der funktionalen Spezifikation	2 ... 5 %
Review der Grobentwurfes	10 ... 16 %
Review des Feinentwurfes	17 ... 22 %
Review des Codes	20 ... 32 %
Modultest	12 ... 17 %
Funktionstest	10 ... 14 %
Komponententest	8 ... 14 %
Systemtest	0,5 ... 7 %

3.2.2 Es wird nicht systematisch bzw. unzureichend getestet

- Ursachen:
  - Es ist angeblich keine Zeit für systematische Tests vorhanden (Termindruck).
  - Die Notwendigkeit für systematische Tests wird nicht erkannt.
  - Die Erstellung von Testspezifikationen für systematische Tests wird nicht entwicklungsbegleitend - beginnend in den ersten Entwicklungsphasen - durchgeführt, sondern erst unmittelbar vor dem erforderlichen Test, so daß der Arbeitsaufwand zum Testzeitpunkt sehr groß ist.
- Folgen:
  - Tests, die nicht systematisch durchgeführt werden, können nicht reproduziert werden, d.h. Regressionstests sind nicht möglich.
  - Verbesserungen der Tests sind nicht möglich, da später auftretende Fehler nicht auf bestimmte (unzureichende) Testschritte zurückgeführt werden können.
  - Nicht entdeckte Fehler
    - können je nach Art der Software hohe Schäden bis hin zu Personenschäden mit allen damit verbundenen Konsequenzen verursachen;
    - verursachen Fehlerbeseitigungskosten, die höher sind als die Kosten, die bei einem systematischen Test sofort nach der Erstellung der Software auftreten.
- Maßnahmen:
  - Die möglichen Konsequenzen unzureichender Tests müssen verdeutlicht werden.
  - In einem Prüfplan müssen die im Entwicklungsablauf durchzuführenden Prüfungen und Tests verbindlich festgelegt werden.
  - Die für Prüfungen und Tests benötigte Zeit muß eingeplant werden.

3.3 Problem- und Systemanalyse

Viele Entwicklungsprojekte scheitern letztendlich daran, daß der Problemanalyse im Vorfeld nicht genügend Beachtung geschenkt worden ist. Fehler, die in dieser Phase gemacht werden, lassen sich später, wenn überhaupt, nur mit sehr hohem Aufwand korrigieren. In dieser Phase wird grob beschrieben, was das zukünftige System leisten soll. Hierzu werden die Methoden zur Funktions- und Datenmodellierung eingesetzt, um Modelle in einer groben Detaillierung zu erstellen. Der Detaillierungsgrad muß eine Aussage über die Wirtschaftlichkeit des geplanten Vorhabens ermöglichen. In dieser Phase gibt es eine enge Zusammenarbeit zwischen Anwendern und Systemanalytikern.

3.3.1 Die Anforderungen und Qualitätsmerkmale werden nicht festgelegt

- Ursachen:
  - Das Pflichtenheft fehlt komplett oder wurde nur mündlich besprochen. Nicht selten wird das Pflichtenheft auch vom Entwickler selbst erstellt.
  - Die Anwender wollen sich noch nicht festlegen oder geben ihre Anforderungen nur unvollständig oder ungenau an.
  - Mangelndes Vorstellungsvermögen der Anwender über die Möglichkeiten der Software-Entwicklung.
  - Die Anforderungen an das System, bzw. seine Leistungsgrenzen werden durch eine "vorläufige" Version der Software ermittelt.
  - Während der Benutzung einer ersten Programmversion ändert sich der Arbeitsstil der Anwender, dadurch ergeben sich für ihn neue Perspektiven und in der Folge auch neue Anforderungen.
  - Wichtige Qualitätskriterien (z.B. das Laufzeitverhalten des Programmes, das Aussehen der Oberfläche, die maximale Anzahl an Einzelaktionen für einen Vorgang) werden nicht spezifiziert.
- Folgen:
  - Die Software muß mit einem erheblichen Mehraufwand geändert oder erweitert werden.
  - Es entsteht Software, die nicht den (nicht schriftlich niedergelegten) Vorstellungen des Kunden entspricht. Bei gravierenden Mängeln aus Anwendersicht kann es passieren, daß die Software gar nicht benutzt wird.
- Maßnahmen:
  - Die Anforderungen der Anwender an die Software und alle wichtigen Qualitätsmerkmale sind schriftlich zu fixieren.
  - Auch vorläufige Entwicklungsziele sind genau festzulegen.
  - Zwischenzeitliche Änderungen am Pflichtenheft müssen möglich sein, jedoch nach einem geregelten Verfahren.
  - Wichtige Qualitätskriterien (siehe oben) müssen schriftlich spezifiziert werden.

3.3.2 Es fehlen eindeutige Begriffsdefinitionen

- Ursachen:
  - Dem Entwickler erscheinen "seine" Begriffsdefinitionen eindeutig und auch für andere Mitarbeiter und den Anwender unmittelbar einleuchtend. Er übersieht dabei die in der Regel unterschiedlichen Sichtweisen und Denkmuster der Anwender und Entwickler.
- Folgen:
  - Erstellen nutzloser Software. Falsch verstandene Sachverhalte führen zu Mißverständnissen und Fehlentwicklungen, was oftmals erst bei der Programmabnahme deutlich wird.
  - Destruktive Diskussionen. Unterschiedliche Vorstellungen der Diskussionspartner über den Diskussionsgegenstand können zu ergebnislosen Diskussionen, oder was noch schlimmer ist, zu weiteren Fehlentscheidungen führen.
- Maßnahmen:
  - Erstellen eines Begriffs-Lexikons, in dem alle verwendeten Fachbegriffe definiert werden. Die Entwickler müssen ihre Arbeit mehr unter dem Gesichtspunkt einer Dienstleistung sehen. Sie sollten sich daher bemühen die "Sprache" der Anwender zu sprechen und Begriffe solange zu hinterfragen, bis alle möglichen Mißverständnisse ausgeräumt sind.
  - Benutzen strukturierter oder objektorientierter Methoden. Dies bietet den enormen Vorteil, daß Begriffe, Strukturen und Abläufe, die im Rahmen der Problemanalyse erarbeitet werden, nahezu ohne Informationsverlust für den Systementwurf und sogar die Codierung weiterverwendet werden können.
  - Vorstellen und Diskutieren von Zwischenergebnissen, sowie gemeinsames Erstellen von Spezifikationen.

3.4 Systementwurf

Der Schritt von der Analyse zur Codierung - die Entwurfsphase - beschäftigt sich hauptsächlich mit folgenden Teilproblemen:

- Sofern noch keine Festlegungen bezüglich der einzusetzenden Hard- und Software getroffen wurden, sind diese auszuwählen.
- Der Funktionsumfang ist zu gliedern und Datenstrukturen sind zu definieren.
- Die Schnittstellen (Datenschnittstellen, Benutzeroberfläche) müssen beschrieben werden.
- Die Managementfunktionalitäten (z.B. Zugriff auf Ressourcen/Daten, Prioritäten im Ablauf, Prozeßsteuerung) sind zu planen.

Weiter zu beachtende Aspekte sind die Wartungsfreundlichkeit, eine eventuelle Wiederverwendbarkeit sowie die Datensicherheit der zu entwickelnden Software. Nach Schätzungen beträgt der Anteil der Wartungskosten etwa 70% der gesamten Entwicklungskosten.

3.4.1 Die Systemarchitektur ist gar nicht oder nur mit großem Aufwand erweiterbar

- Ursachen:
  - Die Gegebenheiten für den Einsatz (Nutzungs-/Lebensdauer, Wiederverwendbarkeit, Ausweitung des Einsatzes der Applikation) werden nicht richtig eingeschätzt.
  - Die Offenheit bei der Verwendung unterschiedlicher Plattformen (Hardware, Standardsoftware) wird nicht berücksichtigt.
  - Häufig fehlt eine kritische Bewertung des Systementwurfs.
  - Es werden keine Entwurfsvarianten vorgeschlagen und diskutiert.
- Folgen:
  - Die Systemarchitektur ist unübersichtlich, was bei Änderungen ein Hauptproblem werden kann.
  - Für Tests entsteht ein erhöhter Aufwand.
  - Die Fehlersuche wird erheblich erschwert.
  - Monolithische Architekturen/Systeme bilden ungünstige Voraussetzungen bezüglich der Pflege und Erweiterbarkeit.
- Maßnahmen:
  - Der Einsatz objektorientierter Methoden und Techniken kann eine übersichtlichere Architektur liefern.
  - Kleinere Programmbausteine lassen neben einer schnelleren Entwicklung auch eine einfachere Fehlerlokalisierung zu.
  - Das Verfeinern des Entwurfs ("vom Groben zum Feinen") wird durch einen objektorientierten Ansatz einfacher, da Detaillierungen und Ergänzungen – basierend auf bereits existierenden Klassen – möglich sind.
  - Ein aktueller Ansatz zur Unterstützung der Designphase ist die Nutzung von Software-Entwurfsmustern: Neben der Problembeschreibung und dem verallgemeinerten Implementierungsproblem wird auch eine Anleitung zum Implementieren gegeben. Durch die Verwendung bewährter Lösungen lassen sich Fehler vermeiden. Verschiedene Unternehmen berichten von einem Anteil der Wiederverwendbarkeit von bis zu ca. 80%, wenn objektorientierte Methoden und Techniken in Design und Implementierung eingesetzt werden [8].

3.4.2 Das System ist nicht modular aufgebaut, die Daten sind nicht gekapselt

- Ursachen:
  - Oft sind die endgültigen Anforderungen zu Beginn einer Entwicklung noch nicht vollständig definiert oder sie werden im Laufe einer Entwicklung modifiziert.
  - Fehlende Kenntnisse oder unzureichende Fähigkeiten der Entwickler im Umgang mit entsprechenden Techniken und Werkzeugen aufgrund fehlender Schulung sind weitere Fehlerpotentiale.
- Folgen:
  - Ein schnelles Entwickeln (Rapid Prototyping) ist nicht möglich.
  - Die Wartung von Systemen wird durch teilweise schwer durchschaubare Abhängigkeiten beeinträchtigt.
  - Daten(strukturen) sind nicht ausreichend geschützt, z.B. vor Mißbrauch, ungewolltem Gebrauch, Verfälschung.
- Maßnahmen:
  - Die Kenntnis und Anwendung entsprechender Methoden und Techniken (u.a. Objektorientierung) unterstützen sowohl eine Modularisierung als auch eine Kapselung. Die jeweiligen Entwickler müssen natürlich die dafür notwendigen Qualifikationen besitzen.
  - Die Entwicklung eines adäquaten Modul- und Datenkonzepts schafft auch für die Implementierung günstige Voraussetzungen.
  - Zum Abschluß der Entwurfsphase ist eine Verifizierung des Modells durch die an der Entwicklung Beteiligten hinsichtlich Vollständigkeit und Eindeutigkeit (Klarheit) durchzuführen.

3.5 Codierung

Die Codierung ist die Umsetzung des Systementwurfs in durch den Rechner interpretierbare "Objekte", z.B. Programme, Module, Makros usw., also in auf dem Computer lauffähigen Code. Man erwartet, daß alle wesentlichen technischen Entscheidungen bereits im Systementwurf festgelegt worden sind, so daß die Implementierung weitgehend mechanisch aus dem Systementwurf herleitbar ist. Die einzelnen Programmodule sollen am Schluß dieser Phase in codierter, dokumentierter und getesteter Form vorliegen. Die Fehler, die in dieser Phase entstehen, führen zu unübersichtlichen Programmen, die keiner warten möchte.

3.5.1 Programmier-Standards bzw. -Richtlinien werden nicht beachtet

- Ursachen
  - Der Sinn von Standards und die damit verbundenen Vorteile sind den Entwicklern nicht klar.
  - Die existierenden Standards sind nicht praxisgerecht und werden deshalb nicht akzeptiert.
  - Es gibt zuviele Standards und Richtlinien. Damit werden sinnvolle Standards durch überflüssige Regelungen "verdeckt".
  - Es existieren gar keine Standards.
- Folgen
  - Die Programme sind schwer zu lesen.
  - Die Wiederverwendung der Software wird erschwert.
  - Bestimmte Mechanismen und Maßnahmen zur Fehlervermeidung, die durch die Standards und Richtlinien implementiert werden, kommen nicht zur Wirkung.
  - Die Portierung auf andere Hardware-Plattformen führt zu kostspieligen Wartungsarbeiten.
  - Eine uneinheitliche und nicht konsistente Benutzerschnittstelle führt zu einer Belastung der Anwender; die Akzeptanz des Programmes schwindet.
  - Die Weiterbearbeitung der Programme durch andere Entwickler wird erschwert.
- Maßnahmen
  - Die Entwickler müssen über die Ziele, die mit der Einführung von Standards und Richtlinien verfolgt werden, ausreichend informiert werden. Eine Einbeziehung der Mitarbeiter bei der Erstellung solcher Richtlinien erhöht die Akzeptanz.
  - Bei jeder Erweiterung von Richtlinien oder Standards sollten die bisherigen erneut auf ihre Notwendigkeit überprüft werden. Auch hierbei ist die Einbeziehung der Mitarbeiter sinnvoll.
  - Die Einhaltung von Standards muß durch ein geeignetes Gremium kontrolliert werden. Dieses sollte nur Mitarbeiter umfassen, die nicht bei der Erstellung der Programme involviert waren.

3.5.2 Die Namensvergabe ist ungünstig

- Ursachen
  - Es existieren keine sinnvollen und verbindlichen Regeln.
  - Die verwendete Programmiersprache erlaubt nur eine eingeschränkte Namenslänge.
  - Nur der Entwickler weiß, was seine Abkürzungen bedeuten (z.B. SNEB = Suchen nach erstem Blank).
- Folgen
  - Namen sind oft nur für den "Erfinder" aussagekräftig und informativ, für andere Mitarbeiter enthält der Name keine sofort erkennbare Information.
  - Die Einarbeitung neuer Mitarbeiter wird erschwert.
  - Der Wartungsaufwand wird erhöht.
- Maßnahmen
  - Zur problemadäquaten Charakterisierung von Typen, Konstanten, Variablen, Prozeduren usw. sollten Namen gewählt werden, die die Funktion dieser Größe bzw. ihre Aufgabe zum Ausdruck bringen. Es sollten Namen vermieden werden, die problemfrei sind oder technische Aspekte, z.B. der Repräsentation bezeichnen.
  - Die Langform des Namens muß als Kommentar angeben werden, wenn mit Programmiersprachen gearbeitet wird, die nur eine eingeschränkte Namenslänge erlauben.
  - Es müssen eigene Richtlinien für die Namensvergabe definiert werden, damit alle gezwungen sind, lesbar zu programmieren. Einheitlichkeit erhöht die Lesbarkeit, das muß jedem klar sein!

3.6 Betrieb und Wartung

Auch für die Zeit nach der Auslieferung von Software an Kunden müssen geeignete Maßnahmen getroffen werden, um die Anforderungen immer komplexerer Software-Produkte zu beherrschen.

Dem Anwender müssen Zugang zum Produkt und dessen Möglichkeiten und Vorteile transparent gemacht werden. Dies erreicht man i.d.R. durch entsprechende Dokumentation und Schulungen.

Die Entwickler müssen darauf vorbereitet sein, u.U. Software-Versionen mit unterschiedlichen Leistungsmerkmalen bereitzustellen, bzw. ältere Software-Versionen weiterzupflegen. Zudem muß zu jeder Software-Version die richtige Dokumentation zur Verfügung stehen. Hierzu bedarf es eines geeigneten Konfigurationsmanagement-Systems.

3.6.1 Die Dokumentation fehlt ganz, ist veraltet oder nicht adäquat

- Ursachen:
  - Fehlendes Problembewußtsein bei Mitarbeitern und Verantwortlichen: Dokumentation ist ohne Stellenwert, sie wird als Luxus angesehen.

- Dokumentation hat keinen Einfluß auf den Gehalt eines Programms. Dies gilt insbesondere für programminterne Kommentare.
- "Geniale" Programmierer halten sich nicht mit dem Schreiben von Dokumentation auf.
- Das Schreiben von Dokumentation ist lästig, insbesondere bei Termindruck. Deshalb wird Dokumentation häufig auf das Notwendigste reduziert und es entstehen Alibi-Dokumente, die niemandem nützen, jedoch den Formalismus befriedigen.
- Die Software wird geändert, aber die entsprechende Dokumentation, vor allem programminterne Kommentare, wird nicht aktualisiert.

- Folgen:
  - Entwicklungsentscheidungen können nur nachvollzogen werden, wenn das Entscheidungsumfeld dokumentiert ist.
  - Schon einmal getroffene Entwicklungsentscheidungen müssen wieder neu erarbeitet werden, falls sie nicht mit den Gründen, die zu der Entscheidung führten, dokumentiert werden.
  - Das Ändern oder Weiterentwickeln von Software gerät langwierig und fehlerträchtig, wenn Programme und Schnittstellen nur unzureichend beschrieben sind. Dies gilt insbesondere, wenn die ursprüngliche Entwicklungsmannschaft nicht mehr vollständig zur Verfügung steht.
  - Der Leistungsumfang von Software ist für den Anwender nicht voll überschaubar, wenn die Leistungsmerkmale nicht komplett oder eindeutig beschrieben sind. Dies kann zu unnötigen Zeitverlusten beim Einarbeiten, Frustration bei den Anwendern und Vertrauensverlust in die Software-Entwickler führen.
  - Außerdem müssen vorhandene, aber unerkannte Leistungsmerkmale möglicherweise durch Zukaufen oder zusätzliches Entwickeln von Software abgedeckt werden.

- Maßnahmen:
  - Die Projektverantwortlichen müssen den Mitarbeitern die Folgen fehlender oder mangelhafter Dokumentation vor Augen halten und so das Problembewußtsein fördern.
  - Für jede Software muß ein zugehöriger Satz von Dokumenten definiert werden.
  - Jedes Dokument muß Minimalanforderungen erfüllen, z.B. eine spezifische, vordefinierte Kapitelstruktur enthalten.
  - Ein Dokumentations-Verantwortlicher muß für alle Unterlagen sein Einverständnis geben.

3.6.2 Die Schulung der Anwender wird vernachlässigt

- Ursachen:
  - Training-on-the-job gilt als ausreichend für "gute" Leute. Deshalb werden Schulungen von Anwenderseite nicht energisch genug gefordert.
  - Der Lernumfang für den Anwender wird vom Entwickler als zu klein eingeschätzt, um dafür eine Schulungsmaßnahme vorzusehen.
  - Zeit oder Kosten für eine Schulung sollen eingespart werden.

- Folgen:
  - Methoden werden nicht richtig eingesetzt und bringen das erwartete Ergebnis deshalb gar nicht oder erst nach Ablauf der geplanten Zeit.
  - Das Einarbeiten in neue Software-Werkzeuge, bzw. Anwendungsprogramme ist langwierig und manche Leistungsmerkmale bleiben unentdeckt. Die daraus resultierende Frustration der Anwender kann zum Imageverlust der Software-Entwicklungsabteilung führen.
  - Frustrierte Anwender zögern mit dem Erteilen weiterer Aufträge; es wird lieber Software gekauft, bzw. woanders in Auftrag gegeben.
  - Die durch das Unterlassen der Schulung eingesparte Zeit wird später durch Fehlbedienungen oder Ineffizienz mehrfach an anderer Stelle verbraucht.
  - Die Anwender-Schulung erfolgt zum falschen Zeitpunkt, zu früh: Informationsverlust beim Anwender, weil er Lerninhalte wieder vergißt; zu spät: verlorene Zeit bei vorherigem Selbststudium.
  - Die Berührungsängste mit neuen Dingen mancher Anwender werden bestätigt und führen so zu längerem Verharren in alten Positionen.

- Maßnahmen:
  - Projektverantwortliche müssen Schulungsmaßnahmen mit Zeitpunkt und Umfang (wenn möglich: im Gespräch mit Anwendern) einplanen.
  - Wirtschaftliche Folgen unterlassener Schulungen unter ganzheitlichem Blickwinkel darlegen. Letztendlich leiden auch die Software-Entwicklungsabteilungen, wenn deren Budget bei Schulungen geschont wird, das gesparte Geld aber an anderer Stelle mehrfach ausgegeben werden muß.
  - Die Vorteile der Neuerungen müssen transparent gemacht werden, um die Motivation der Anwender zur Innovation zu erhöhen.

3.6.3 Das Konfigurationsmanagement ist unzureichend

- Ursachen:
  - Termindruck: in der Eile gibt man sich damit zufrieden, alles im Kopf zu haben.
  - Das Unterschätzen des Nutzerkreises, der Lebensdauer oder des Umfanges der zu entwickelnden Software. Vermeintlich wenige Nutzer, Interimslösungen oder kleine Programme lassen ein Programm gefühlsmäßig leichter beherrschbar erscheinen. Deshalb wird das etwas aufwendigere Konfigurieren mit gutem Gewissen unterlassen.
  - Das Unterschätzen der Personalfluktuation: man ist der Ansicht, die jeweiligen Experten immer verfügbar zu haben.
- Folgen:
  - Das Altern der Software wird wegen des fehlenden Überblickes beschleunigt. Mit der Zeit erfährt fast jedes Programm Änderungen, die zu Beginn nicht vorhersehbar waren und somit meist "irgendwie" in die bestehende Programmstruktur eingebracht werden.
  - Ohne ein klares Konfigurationsmanagement verliert man schnell die Kontrolle, welcher Kunde welche Module in welcher Version ausgeliefert bekam.
  - Die ausgelieferte Software neigt zu Fehlern, weil alte Module verwendet wurden. Entweder wurden die richtigen Module nicht berücksichtigt oder sogar durch alte Module überschrieben.
  - Das Pflegen der Software führt zu "Verschlimmbesserungen". Es kann passieren, daß mit Fehlerkorrekturen oder Weiterentwicklungen betraute Mitarbeiter veraltete oder als Testversion gedachte Module als die aktuell gültige Programmversion ansehen und so Veränderungen schaffen, die möglicherweise mit den tatsächlich gültigen Modulen nicht mehr korrespondieren. Oder es werden nicht mehr aktuelle Module korrigiert, die schon von einer weiteren Person in anderer Weise verändert wurden. Das schlußendliche Chaos kann im Extremfall nur durch die komplette Neuentwicklung der Software bereinigt werden.
- Maßnahmen:
  - Projektverantwortliche müssen das Problembewußtsein bei allen Beteiligten herbeiführen.
  - Es müssen geeignete Software-Konfigurationsmanagement-Tools bereitgestellt werden.
  - Es sollten projektbezogen Konfigurationsmanagement-Verantwortliche ernannt werden, ohne deren Einverständnis keine Software ausgeliefert werden darf.

4 Zusammenfassung

Wie anfangs bereits erwähnt, müssen viele der beschriebenen Fehler und Probleme immer in Zusammenhang mit den jeweiligen Unternehmensstrukturen gesehen werden. Ein allgemeingültiges Lösungskonzept gibt es nicht. Vielmehr gilt es, in einer Gratwanderung zwischen Reglementierungen und Vorschriften einerseits sowie Freiräumen für die individuelle Kreativität andererseits in der Praxis den besten Mittelweg zu finden. Es bleibt jedoch ein schwieriges Unterfangen, alle Personen zufriedenzustellen, die an der Software-Entwicklung direkt und indirekt beteiligt sind: Entwickler, Anwender und Management.

Literatur

[1] Beims, H.D.: Praktisches Software Engineering, Hanser Verlag, 1995

[2] Davis, A.D.: 201 Principles of Software Development, McGraw-Hill, 1995

[3] DeMarco, T.: Software-Projektmanagement, Wolfram's Fachverlag, 1989

[4] Denert, E.: Software-Engineering, Springer-Verlag, 1992

[5] Frick, A.: Der Software-Entwicklungsprozeß - Ganzheitliche Sicht, Hanser Verlag, 1995

[6] Frühauf, K.; Ludewig J.; Sandmayr H.: Software-Projektmanagement und -Qualitätssicherung, vdf Verlag der Fachvereine Zürich, 1988

[7] Krüger, M.: Prüfen von Software – aber richtig, Elektronik Nr. 24, 1993

[8] Martin, J.; Odell, J. : Object Oriented Analysis and Design, Prentice Hall 1992

[9] Myers, Glenford J.: Methodisches Testen von Programmen, 4.Auflage, Oldenbourg, 1991

[10] Schäfer, S.: Objektorientierte Entwurfsmethoden, Addison-Wesley, 1994

[11] Wallmüller, E.: Software-Qualitätssicherung in der Praxis, Hanser, 1990