12 Ansätze zur Beurteilung / Verbesserung der Produkt- und Prozessqualität

Dieses Kapitel befasst sich mit Ansätzen zur Beurteilung und Verbesserung der Produkt- und Prozessqualität, die einerseits ein gutes Umfeld für das Testen von Software schaffen und andererseits die Testaktivitäten in der Entwicklung ergänzen können und sollen. Das Kapitel soll Projekt- und Qualitätsmanagern die Grundlagen für die Auswahl und Verwendung derartiger Ansätze vermitteln.

Flexible Ansätze für die Entwicklung. Veränderte Anforderungen an Softwareprojekte und -produkte erfordern eine Anpassung der Vorgehensmodelle und deren Entwicklungsmethoden. Durch kürzere Entwicklungszeiten und entsprechend engere Kundenbeziehungen entstehen neue Ansätze in der Produktentwicklung. Zusätzlich notwendige Flexibilität kann beispielsweise durch den Einsatz so genannter Agiler Ansätze [Agile Alliance, 2003] erreicht werden. Agile Ansätze folgen dem Grundsatz, in sehr kurzen Entwicklungszyklen dem Kunden ein lauffähiges System zu präsentieren. Entsprechend sind die Qualitätsmanagementmethoden und Testansätze der traditionellen Entwicklungsprozesse an dieses Konzept anzupassen.

Workshops zur Anforderungserfassung. Anforderungen variieren typischerweise nach Anwendungsgebiet, Projektart und den Projektbeteiligten. Die Sichten von Anwendergruppen in einem Projekt können untereinander beträchtlich variieren und unterscheiden sich oft stark von der Sichtweise eines Projektleiters oder Entwicklers. Entsprechend gilt es, diese Anforderungen vorab klar zu definieren und zu verhandeln, um ein möglichst breites Fundament für eine „WinWin"-Situation, also einen breiten Konsens, zu erreichen. Groupware-Systeme, wie EasyWinWin [EWW, 2003] können diesen Verhandlungsprozess effektiv strukturieren und unterstützen. Gut verhandelte, im Kern stabile Anforderungen sind eine wesentliche Grundlage für solide Qualitäts- und Testplanung.

Frühzeitige Reviews. Erfahrungsgemäß treten in jedem Projekt während aller Phasen des Entwicklungsprozesses Fehler auf. Fehler können zu erheblichen Aufwänden für unproduktive Nachbearbeitungstätigkeiten führen, sofern sie nicht frühzeitig erkannt und behoben werden. Die Bandbreite reicht vom einfachen Korrigieren eines Tippfehlers (trivialer Fehler) über Anpassungen im Design (leichte und schwere Fehler) bis zur umfassenden Neukonzeption des Projekts (bei kritischen Fehlern). Einen wichtigen Beitrag zur frühzeitigen Fehlererkennung leisten auf Projektebene Reviews und Inspektionen. Diese Ansätze können im Gegensatz zum Testen schon eingesetzt werden, sobald ein Zwischenprodukt existiert, auch wenn dieses nicht lauffähig ist.

Ansätze zur Prozessbeurteilung und -verbesserung, wie etwa Audits und Assessments und verschiedene Modelle zur kontinuierlichen Verbesserung wie das EFQM Modell, helfen auf Unternehmensebene effektive und effiziente Entwicklungsprozesse herzustellen. Gut abgestimmte Entwicklungsprozesse unterstützen eine qualitativ hochwertige Softwareherstellung und schaffen wesentliche Voraussetzungen für erfolgreiches Testen.

12.1 Agile Ansätze der Softwareentwicklung (Flexibilität)

Agile Ansätze in der Softwareentwicklung werden, speziell bei relativ kleinen Projekten als alternatives Vorgehensmodell zu herkömmlichen Modellen, wie beispielsweise V-Modell oder Spiralmodell eingesetzt. Durch die sehr kurzen Iterationen in den jeweiligen Phasen des Entwicklungsprozesses ist ein erhöhter Kommunikationsaufwand und verstärktes Einbinden des Kunden notwendig.

Agil heißt – nach Hruschka [Starke, 2002] – beweglich und flexibel zu sein und auf neue Anforderungen mit neuen Lösungsansätzen zu reagieren und nicht zu lange an alten Dogmen festzuhalten oder „Dienst nach Vorschrift“ zu verrichten. Ein agiles Vorgehen beurteilt zusätzlich das jeweilige Risiko der verschiedenen Aufgaben bei der Softwareentwicklung und wählt dann die geeigneten Maßnahmen. Dabei werden folgende Schwerpunkte gesetzt:

- Offenheit für Änderungen, statt Festhalten an alten Plänen
- Ergebnisorientiert statt prozessorientiert
- Kommunikation: „miteinander darüber reden“ statt „gegeneinander schreiben“
- Mehr Vertrauen, weniger Kontrolle

Bewährte Ansätze, sog. "*Best Practices*", sollen unter den Mitarbeitern ausgetauscht und etabliert werden, anstatt Vorgaben von oben zu diktieren

Trotz allem heißt „Agilität“ nicht, dass Chaos entstehen muss:

- Agile Vorgehensmodelle haben Ergebnisse. Diese unterscheiden sich aber für unterschiedliche Projekte in Anzahl, Tiefgang und Formalismus
- Auch agile Entwicklung kennt Prozesse, nur lassen diese mehr Spielraum für lokale Alternativen und Kreativität
- Agile Methoden setzen auf Verantwortung; es werden jedoch nur notwendige Rollen besetzt

Das Risiko ist der entscheidende Faktor: jeder Projektleiter, Systemanalytiker, Architekt, Tester und Designer überprüft ständig die vorhandenen Risiken und entscheidet in seinem Umfeld über die notwendigen Maßnahmen, um zu verhindern, dass aus Risiken Probleme werden.

Fehler in der Softwareentwicklung sind teuer, es gilt also Fehler zu vermeiden. Traditionelle Prozesse erreichen dieses Ziel mit Dokumentation und Redundanz. Die agile Entwicklung hingegen basiert auf der Idee, dass laufende Systeme besser zu beurteilen sind als umfangreiche Dokumentation.

12.1.1 Das agile Manifest [Manifest, 2003]

Im Februar 2001 trafen sich 17 Entwickler und formulierten gemeinsam das „*Manifesto for Agile Software development*“. Sie einigten sich auf die, in der Definition von Hruschka genannten Prinzipien und nannten sich die „*Allianz für agile Softwareentwicklung*“ [Agile Alliance, 2003].

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

Agiler Ansatz	**Konventioneller Ansatz**
Einzelpersonen und Interaktionen	Prozesse und Werkzeuge
Laufende Software	Umfangreiche Dokumentation
Zusammenarbeit mit dem Kunden	Vertragsverhandlungen
Flexibilität	Anhalten an strikten Plänen

Tabelle 12.1: Verleich „Agiler Ansatz“ versus „Konventioneller Ansatz“

That is, while there is value in the items on the right, we value the items on the left more. This group is about helping, not telling. The Alliance members want to help others with agile methods, and to further our own knowledge by learning from those we try to help.” [Martin Fowler]“.

Die Allianz erkennt zwar die Schwerpunkte der herkömmlichen Softwareentwicklung (rechte Seite) an, misst allerdings den „neuen“ Paradigmen (linke Seite) eine höhere Bedeutung zu. Im Detail kann der zitierte Auszug aus dem „Agilen Manifest“ folgendermaßen interpretiert werden:

“Einzelpersonen und Interaktionen” gegen “Prozesse und Werkzeuge”: Prozesse sind wichtig für effiziente Abwicklung, aber sie ersetzen weder Können, Ausbildung noch das „miteinander Reden“, da dies oft die effizienteste Form der Kommunikation darstellt.

“Laufende Software” gegen “Umfangreiche Dokumentation”: Dokumentation hilft bei der Festlegung von Anforderungen und der gemeinsamen Kommunikation, ist aber trotzdem nur ein Modell; die Software ist die Realität ("Die Software muss das Geld verdienen, nicht die Dokumentation" [Coldewey Consulting, 2001]).

“Zusammenarbeit mit dem Kunden” gegen “Vertragsverhandlungen”: Verträge sind wichtig, damit jeder weiß, was vereinbart worden ist, und um beim Scheitern Verantwortung zu teilen, aber nach einem großen Streit mit dem Kunden oder gar einer Gerichtsverhandlung wird das Projekt kaum noch Erfolg haben. Deswegen wird ständige Kommunikation mit dem Kunden als wichtiger und zielgerechter Ansatz angesehen.

“Flexibilität” gegen “Anhalten an strikten Plänen”: Planung wird benötigt um Kosten abzuschätzen, jedoch muss genügend Flexibilität vorhanden sein, da sich die Prioritäten und Anforderungen meist im Laufe des Projekts ändern.

Die Anwendung agiler Methoden erfolgt größtenteils in kleineren Projekten; für den Einsatz in größerem Projektumfeld sind weitere Forschungen notwendig, insbesondere in Hinblick auf frühe Leistungsvereinbarungen und Preisfindung zwischen Anbieter und Kunden.

12.1.2 Das agile Projekt

Das agile Projekt wird durch folgende Schwerpunkte gekennzeichnet:

Zusammenarbeit mit dem Kunden: Das INCOSE Handbuch [Polen, 2002] beschreibt die Phasen des Lebenszyklus, beginnend mit der Anforderungsanalyse, und führt über Konzeptforschung und Erstellen von Prototypen (*Prototyping*) zur Entwicklung und Produktion. Am wichtigsten ist es, möglichst viel Information über die Wünsche und Bedürfnisse des Kunden und über alternative Lösungen zu generieren, um optimale Lösungen bei geringem Risiko planen zu können.

Lauffähige Software so früh wie möglich: Es muss bedacht werden, dass Zeit der kostbarste Faktor ist. Solange das System nicht ausgeliefert ist, hat der Kunde keinen Nutzen. Es kann unvermeidbar werden, temporäre Alternativen zu entwerfen und zu planen. Das wiederum führt zu komplexeren, teureren, und langsameren Projekten.

Dokumentation: Die herkömmliche formale Dokumentation verliert stark an Bedeutung und wird durch die inkrementelle Vorgangsweise, unter ständiger Miteinbeziehung des Kunden, ersetzt. Die Spezialisierung auf gewisse Modellierungstechniken wird zurückgedrängt. Es wird an allen Modellarten gleichzeitig gearbeitet und nicht zuerst ein Anwenderfallmodell entworfen und dann erst das Datenmodell. Es wird sehr viel Wert darauf gelegt, dass man weiß, wann man mit welchem Diagramm welche Ziele erreichen kann. Es gibt Zeitpunkte, zu denen ein Modell nicht ausreichend ist, daher ist es manchmal nötig, mehrere Modelle parallel für dasselbe System zu erstellen. Die Modelle sollen nur aktualisiert werden, wenn dies unbedingt notwendig ist. Es kostet zu viel Zeit und Geld die Dokumentation immer an den Quellcode anzupassen, diese Ressourcen werden bei der Entwicklung benötigt.

Überschaubarkeit durch Einfachheit: Die Idee, das System einfach zu halten spiegelt sich auch hier wieder. Modelle sollen in einfacher Umgebung erstellt werden, meistens reicht sogar ein Blatt Papier. Es soll alles Unnötige beiseite gelassen werden, gleichzeitig müssen aber alle Anforderungen erfüllt werden. Die Modelle dienen nur dem besseren Verständnis und können beiseite gelassen werden, sobald alle die Anforderungen verstanden haben.

12.1.3 Agile Methoden

Agile Projekte erfordern den Einsatz von entsprechenden agilen Methoden, die von folgenden Voraussetzungen ausgehen bzw. folgende Ziele verfolgen:

- Verringerung des Aufwandes für Pläne, Analysen und der Dokumentation im Allgemeinen, Abweichungen von diesem Plan werden als Zeichen weiterer Erkenntnisse gehandhabt.
- Kleine Teams - "Wer zusammen entwickelt muss zusammen sitzen" [Coldewey Consulting, 2001]
- Inkrementelle Softwareentwicklung, mit klaren Zielen jeder Phase
- Termine dürfen nicht verschoben werden, stattdessen wird der Lieferumfang angepasst.
- Überleben in chaotischer Umgebung
- Konzentration auf Ergebnisse als gemeinsam getragene Vision, nicht auf Einhalten einer bestimmten Methodik

Aus diesen Voraussetzungen wurden einige Ansätze für agile Methoden entwickelt bzw. abgeleitet. Die nächsten Abschnitte zeigen XP als beispielhaften agilen Prozess und weitere wichtige Methoden im Überblick.

eXtreme Programming (XP)

Die Innovationsgeschwindigkeit in der Informationstechnik bewirkt, dass ein Festhalten an starren Prozessen immer schwerer wird, wenn ein Unternehmen am Markt konkurrenzfähig bleiben will.

Hier knüpft die Grundidee von Extreme Programming an, es geht darum, dem Kunden so schnell wie möglich laufende Software zu liefern und möglichst flexibel auf geänderte Anforderungen reagieren zu können

XP hat eine Reihe von Prinzipien, die den Zugang verdeutlichen:

- *Einfachheit*: Bei der Implementierung soll die möglichst einfachste aller Lösungen für ein Problem genutzt werden. Es wird absichtlich nicht an Änderbarkeit zu späteren Zeitpunkten gedacht, möglichst einfach muss es sein.
- *Kurze Iterationen*: Eine Iteration dauert zwischen ein und drei Wochen.
- *Kommunikation mit Kunden und Kollegen*: Der Auftraggeber muss in den Prozess einbezogen werden, er muss die Software so früh wie möglich in Anbetracht nehmen können, um neue Anforderungen definieren zu können oder Missverständnisse aufzuklären, denn späte Änderungen sind teuer.
- *„Programmierung in Paaren“*: Die Programmierung erfolgt meistens in Zweierteams, wobei abwechselnd einer den Code schreibt und sich der andere Lösungen für aktuelle Probleme überlegt. Durch diese direkte Kommunikation entstehen weniger Missverständnisse, beide profitieren und haben mehr Spaß an der Arbeit.
- *Automatisierte Tests*: Die Programmierung beginnt mit der Erstellung eines Tests. Erst dann wird mit der Implementierung der eigentlichen Aufgabenstellung begonnen. Durch diese Methodik, so ist es gedacht, soll genau das programmiert werden, was nötig ist um den Test erfolgreich zu absolvieren und nicht mehr.

XP beginnt mit der Entwicklung sogenannter *User Stories*, siehe Abbildung 12.1, die denselben Zweck haben wie *Use Cases* (Anwendungsfälle). User Stories werden vom Kunden erstellt und enthalten eine Beschreibung der Anforderungen an das zu erstellende System, geschrieben in drei bis vier Sätzen und in einfachen Worten gehalten. Immer dann, wenn einzelne Anforderungen unklar erscheinen wird ein *Architectural Spike* entwickelt, ein auf das einzelne Problem konzentrierter rudimentärer Prototyp. Spikes sind Codeexperimente bzw. Lösungen, deren Zweck es ist herauszufinden, wie und ob man ein bestimmtes Problem lösen kann.

Auf Basis von User Stories und Architektural Spikes erfolgt das *Release Planning*, das den zentralen Bestandteil des XP darstellt. Hier wird das Expertenwissen über die Anwendung des Kunden zusammen mit dem Expertenwissen über die Entwicklung der Programmierer zur Planung der Funktionalität des nächsten Prototypen genutzt. Hier werden auch die Teams zu den einzelnen User Stories zugeordnet.

Mit Hilfe des entstandenen Release Plan wird die nächste Version des Systems codiert und anschließend getestet. Je nach Ergebnis der Tests wird entweder eine neue Iteration des Release Plannings oder ein neuer Codierzyklus angestoßen oder ein Zwischenergebnis (*Small Release*) erzeugt.

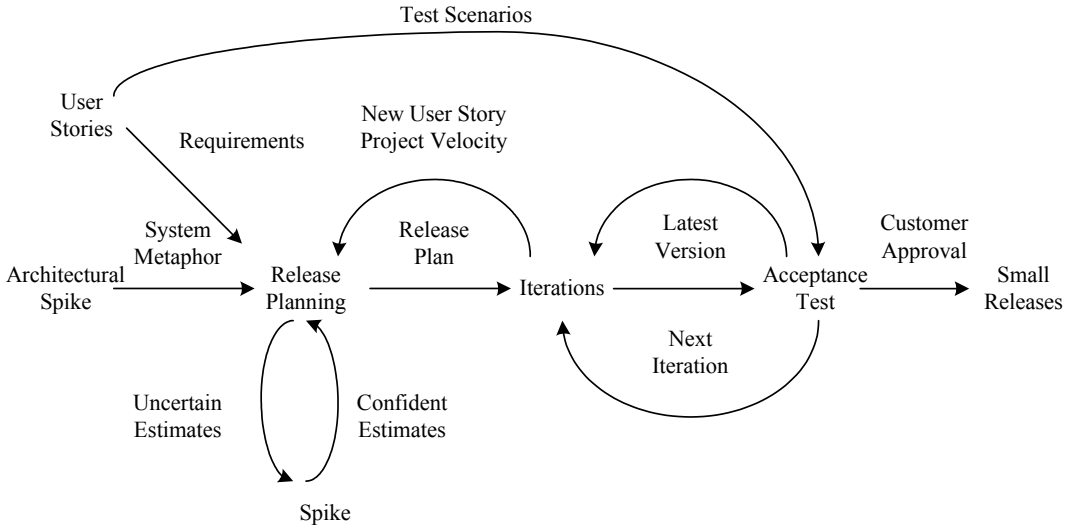


Abbildung 12.1: XP-Projektablauf [XP, 2002]

Der Entwicklungsprozess wird durch die Verwendung sogenannter *Best Practices* (die Anwendung von bewährten Vorgehensweisen) unterstützt. Beispiele dafür sind das Programmieren in Paaren und das Refactoring, das eine kontinuierliche Veränderung und Restrukturierung zwecks Einfachheit des Codes ohne Veränderung des Systemverhaltens unterstützen soll.

Üblicherweise werden Methodiken, wie XP, dann angewandt, wenn ein Projekt schon in Schwierigkeiten ist. Die Problemstellen werden identifiziert und ein geeigneter Ansatz aus XP ausgewählt. Bei häufig auftretenden Änderungen der Anforderungen sind User Stories und kurze Iterationen empfehlenswert. Wenn eines der Probleme die hohe Anzahl an Bugs ist, dann können automatisierte Tests zur Lösung beitragen.

Dieses Vorgehen sollte fortgesetzt werden, solange bis keine Probleme mehr übrig sind. Danach sollten all die restlichen XP-Praktiken in den Entwicklungsprozess integriert werden.

Alternative agile Methoden im Überblick

Eine Grundidee von *Scrum* [Scrum, 2003] basiert auf der Vorgabe, weniger Zeit mit der Erstellung von Plänen, Dokumentationen und Managementberichten zu "verschwenden" und stattdessen die Teamarbeit mit Fokus auf die kurzfristigen Ziele verstärkt einzusetzen. Am Beginn einer Iteration (*Sprint*) werden die Merkmale (features) in so genannten *Backlogs* festgelegt und nach Beendigung eines Sprints, der üblicherweise eine Dauer von ca. 30 Tagen aufweist, überprüft und angepasst. Teaminterne Treffen werden alle 2-3 Tage unter der Anleitung eines so genannten *Scrum*

Masters durchgeführt, bei dem die kurzfristigen Ziele und Änderungen definiert werden. Durch diese kurzen Zeitabstände zwischen den teaminternen Treffen ist es möglich, auf geänderte Anforderungen sehr schnell reagieren zu können.

Die *Crystal Family* [Crystal, 2003], die auf Alistair Cockburn zurückgeht, stellt je nach Projektart und -größe eine Auswahl von geeigneten Methoden zur Verfügung. Derzeit wird zwischen vier Methoden, die auf die Projektgröße abgebildet werden, unterschieden. Der Ablauf ist jedoch nicht vollständig festgelegt sondern primär durch einige Grundeigenschaften definiert. Die erste Iteration wird durchgeführt, um nach Abschluss dieser Iteration aus den gemachten Fehlern zu lernen und in folgenden Zyklen zu verbessern. Am Ende eines jeden Zyklus erfolgt unter Berücksichtigung der neu gewonnenen Erkenntnisse die Planung für den weiteren Projektverlauf.

Bei traditionellen Methoden ist die Funktionalität des Endproduktes fix, Zeit und Ressourcen sind variabel. Bei *DSDM* (Dynamic Systems Development Method) [DSDM, 2003] sind Zeit und Ressourcen die fixen Größen und die Funktionalität ist variabel. Auch hier gibt es Iterationen, so genannte *Time Boxes*. Die Time Boxes sollten so bemessen sein, dass sie gut und realistisch planbar sind. Normalerweise dauert eine Time Box einige Tage bis wenige Wochen.

Der Prozess besteht aus fünf Phasen, die in folgender Reihenfolge absolviert werden:

1. *Machbarkeitsstudie*: Durch die Machbarkeitsstudie wird die Anwendbarkeit von DSDM im aktuellen Projekt überprüft. Weiter erfolgt in dieser Phase die Definition des Problems, die Schätzung der Kosten und die technische Realisierbarkeit des Projektvorhabens.

2. *Geschäftsstudie*: Der Fokus der Geschäftsstudie liegt auf den Geschäftsprozessen und weiteren Informationen, die durch das zu entwickelnde System abgedeckt werden. Es gilt, in kollaborativen Workshops mit erfahrenen Mitarbeitern, eine sogenannte "Business Area Definition" zu erarbeiten, die oben genannte Informationen enthält und zusätzlich die Benutzer des zukünftigen Systems identifiziert.

3. *Funktionales Modell*: Die Ergebnisse der vorherigen Phase werden überarbeitet und erweitert und erste Prototypen werden erstellt und getestet.

4. *Design & Build*: Hier werden die Prototypen in vier Iterationen implementiert und getestet, das Endprodukt dieser Phase ist ein getestetes System.

5. *Implementierung*: Unter dieser Phase wird bei DSDM, ganz im Gegensatz zu traditionellen Methoden, der Übergang des Produkts vom Entwicklungsstadium in das operative Stadium verstanden.

Die Implementierung und der Test wird im traditionellen Sinn durchgeführt und erfolgt hauptsächlich in den Phasen 3 und 4. Zusätzlich gibt es noch eine Pre- und eine Postprojektphase.

Der Fokus des *Adaptive Software Development* (ASD) [Highsmith, 2000; ASD, 2003] liegt hauptsachlich auf der Erstellung von großen, komplexen Softwaresystemen. Der Begründer dieser Methodik, Jim Highsmith, sieht das Erstellen von und das Festhalten an Plänen als paradox an, da der Ausgang immer ungewiss ist. In klassischen Prozessen werden Abweichungen von Plänen als Fehler angesehen, in einer adaptiven Umgebung jedoch, wird es als ein Schritt in Richtung der korrekten Lösung gesehen.

In einer adaptiven Umgebung, in der alle Beteiligten lernen und neue Erfahrungen machen, fließen die Erkenntnisse aus vergangenen Iterationen in zukünftige Iterationen und Projekte ein. Die Anpassung von Plänen und Design an die neu gewonnen Erkenntnisse, sind eine logische Konsequenz aus diesem Ansatz. Die Software wird schrittweise unter der Verwendung von Prototypen erarbeitet, eine Iteration ist mit ungefähr vier Wochen bemessen. Das Hauptaugenmerk liegt auf den Ergebnissen und deren Qualität, nicht auf der Qualität der Entwicklungsschritte.

12.1.4 Vorteile des agilen Konzepts

Bei der traditionellen Art der Entwicklung müssen alle Anforderungen bereits vor dem Start der Entwicklung feststehen und jede Änderung bedingt eine Verzögerung des Entwicklungsprozesses. Agile Entwicklung ermöglicht durch iterativen Charakter und die ständige Einbeziehung des Kunden, dass der Kunde nicht von vornherein die Anforderungen strikt überlegen muss, bevor er das System überhaupt noch "gefühlt" hat.

Die Vorteile sind also:

- *Flexibilität*: Die Anforderungen müssen nicht von Anfang an feststehen.
- *Kundenfokus*: Der Kunde ist in den Informationsfluss integriert, alle wichtigen Entscheidungen werden gemeinsam getroffen.
- *Minimalismus*: Dokumente, Pläne, Analysen werden auf ein Minimum begrenzt. Dadurch wird Überschaubarkeit gesteigert. Weiters steht mehr Zeit für die Implementierung zur Verfügung.
- *Kollaboration*: Mitarbeiter kommunizieren direkt miteinander (face-to-face), wodurch die Kommunikationsbreite gesteigert und die Wahrscheinlichkeit von Missverständnissen herabsetzt werden kann.
- *Software*: Es wird versucht, dem Kunden bereits sehr früh Prototypen zu liefern, damit dieser eine Vorstellung davon bekommt wie sein Produkt aussehen wird und neue Anforderungen formulieren kann.

12.1.5 Nachteile des agilen Konzepts

In gewissen Situationen, in welchen die Grundvorrausetzungen für agile Entwicklung nicht oder nur teilweise gegeben sind, können die Methoden nur beschränkt angewandt werden:

- *Aufwand der Kommunikation*: Wenn Mitarbeiter und Kunden geografisch getrennt sind, kann die von agilen Methoden geforderte Vier-Augen Kommunikation nicht durchgeführt werden. Abhilfe schaffen hierbei regelmäßige Treffen.
- *Granularität:* Große Teams können nur schwer agile Methoden verwenden, da die Verwaltung der großen Anzahl von Kommunikationslinien die Effizienz der Methoden sinken lässt. Agile Methoden können trotzdem angewandt werden, der Agilitätsgrad wird aber gering sein.

- *Erschwerte Wiederverwendung:* Agile Methoden sind auf die Lösung eines bestimmten Problems fokussiert, die Wiederverwendung von Komponenten wird daher erschwert. Die Adaptierung von agilen Methoden auf andere Problemfelder, Projekte, Anwendungsbereiche ist nicht einfach, da entsprechende Richtlinien bzw. Vorgehensweisen (noch) unklar sind.
- *Flexibilität zulasten formaler Kriterien:* Sicherheitskritische Anwendungen sind Anwendungen, bei denen ein Fehler Menschen verletzen oder schwere wirtschaftliche Einbußen herbeiführen können. Traditionelle Techniken (formelle Spezifikation, rigoroses Testen, …) sind in einem solchen Umfeld kalkulierbarer, aber auch teuerer. Der sinnvollste Weg ist der Einsatz von formellen Techniken unter Berücksichtigung der agilen Methoden.

12.2 Strategische Definition von Anforderungen

Qualität im Softwareprojekt ist maßgeblich abhängig von der Anforderungsspezifikation, der Rate der Anforderungsänderungen, und der Managementreaktion auf solche Anforderungsänderungen.

Dieses Kapitel erläutert einen aktuellen Ansatz für das Erstellen einer Anforderungsdefinition, die solide Basis für ein Softwareprojekt sein kann. Groupware-Systeme können die Qualität der Kommunikation zwischen den erfolgskritischen Teilnehmern insbesondere in großen Projekten signifikant erhöhen. Der EasyWinWin-Ansatz (EWW) hilft, eine Gewinnsituation für alle beteiligten Rollen zu bestimmen bzw. wesentliche Zielkonflikte und Risiken offen zu legen.

12.2.1 Anforderungsdefinition

Ein unzulängliches Erfassen von Anforderungen lässt viele Softwareprojekte bereits in der Anfangsphase scheitern. Die Ergänzung bzw. Korrektur dieser fehlenden, unklaren und fehlerhaften Anforderungen ist in späteren Entwicklungsphasen meist nur durch schwierige und oft kostenintensive Nachbesserungen möglich. Abhängig von der Fehlerart (*critical defects*) bzw. deren Auswirkung ist unter Umständen keine Korrektur mehr möglich.

So zeigen auch Umfragen, dass 30% der Projekte vorzeitig scheitern und es in 50% der Fälle Probleme mit dem Anforderungsmanagement gibt. Ein häufiger Grund ist die oft unstrukturierte Vorgehensweise bei gemeinsamen Erhebungen, Abklärungen und Verhandlungen von Anforderungen.

Das Bestimmen von Anforderungen in größeren Projekten ist üblicherweise ein komplexer und schwieriger Prozess. Zum einen gehören zu einem Softwareprojekt meist mehrere erfolgskritische Projektbeteiligte (so genannte *stakeholders*). Diese verfügen über unterschiedliche Erwartungen und Standpunkte, die kontrolliert in das Projekt einfließen sollen. Zum anderen ist die exakte Definition einer Anforderung ein oft zeitaufwendiger Vorgang, der sich aber auf jeden Fall lohnt.

12.2.2 Win-Win und Win-Lose

Die aus der Spieltheorie kommende Bezeichnung Win-Win definiert Bedingungen zwischen zwei oder mehreren Beteiligten, bei denen jeder gewinnen kann. So ist es auch in Softwareprojekten als Ziel eine Win-Win Situation anzustreben. Durch verschiedene Verhaltensmuster der einzelnen Menschen bzw. Parteien innerhalb des Teams ist es möglich, dass sich alle Beteiligten in einer der

Situation eines Gewinners befinden können. So bekommt der Kunde beispielsweise das fertig entwickelte Programm und die Entwicklungsfirma dafür eine angemessene finanzielle Entlohnung. Zudem stärkt eine Win-Win Situation die Geschäftsbeziehung zwischen den einzelnen Projektbeteiligten.

Eine Win-Lose Beziehung, bei der die eine Seite auf Kosten der anderen Seite mit einem Gewinn aussteigt, ist zwar für die eine Seite lukrativ, verändert sich aber meistens in eine Lose-Lose Beziehung. So gewinnt beispielsweise mit einem schnell, dafür aber schlampig programmierten Programm die Entwicklerfirma, langfristig gesehen wird aber die Firma einen schlechten Ruf bekommen und schlussendlich verlieren beide Seiten.

12.2.3 Systeme mit Groupwareunterstützung

Eine wesentliche Komponente für die Schaffung einer Win-Win Situation ist das Schaffen einer guten Kommunikationsbasis zwischen den Beteiligten. Die direkte Kontaktaufnahme in einer Besprechung ist oft zeitaufwendig und ineffizient.Elektronischen Kommunikationstechnologien, so genannte Groupwaresysteme, unterstützen Gruppen bei der effektiven und effizienten Kommunikation. Derartige Systeme haben i.a. einen gemeinsamen Kalender, gemeinsam genutzten Speicherplatz für Dokumente und eine erleichterte E-Mail Kommunikation. Das heißt, jedes Teammitglied kann einfach Daten und Informationen für alle anderen Personen im Team publizieren. Groupwaresysteme unterstützen die Teammitglieder bei ihren teaminternen Aufgaben. Der Einsatz des Werkzeugs steht nicht im Mittelpunkt sondern wird als unterstützendes Hilfsmittel gesehen und angewandt. Für die verschiedenen Aufgaben, wie beispielsweise das Finden von Anforderungen stehen innerhalb von Groupwaresystemen Werkzeuge für Brainstorming, Bewertung, usw. zur Verfügung.

12.2.4 EasyWinWin

Die auf Basis eines Groupwaresystems entwickelte Softwareprozess EasyWinWin ist ein Verfahren zur systematischen Erhebung und Verhandlung von Anforderungen in Softwareprojekten. Damit können die unterschiedlichen Erwartungen in einem Projektteam analysiert werden, um Win-Lose oder gar Lose-Lose Situationen zu vermeiden.

Die Vorgehensweise zur Schaffung einer Gewinnbedingung kann durch das EWW Modell dargestellt werden (Abbildung 12.2). Im ersten Schritt werden Win-Conditions pro Rolle (Stakeholder) gesammelt. Aufbauend auf diesen "idealen" Anforderungen erfolgt die Ausarbeitung wichtiger rollenspezifischer Konflikte und Problemkreise (*Issues*) und die Sammlung möglicher Lösungen bzw. Alternativen (*Options*) um diese Konflikte bzw. Probleme zu lösen. Ein gefundener Konsens führt zu einer allgemeinen Zustimmung (*agreement*). Win-Conditions ohne Issues werden ebenfalls automatisch als Zustimmung gewertet, da von keinem Beteiligten Problemfelder lokalisiert wurden. Win-Conditions mit *offenen Issues* werden als Risiken eingestuft und entsprechend dokumentiert.

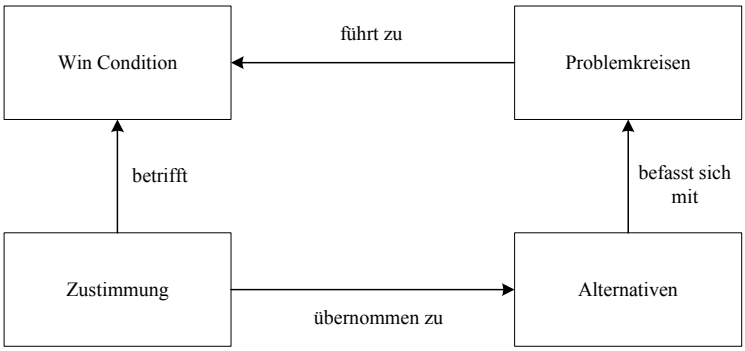


Abbildung 12.2: EWW Modell

Während des EWW-Prozesses bewerten die Teilnehmer für jede Win-Condition den Geschäftswert und die Schwierigkeit der Umsetzung. Entsprechend werden die Win-Conditions sortiert, und die besonders interessanten vorrangig weiter analysiert. Das Abstimmungsverhalten der Teilnehmer zeigt auch auf, bei welchen Win-Conditions Konsens über Nutzen und Umsetzung herrscht bzw. wo es offensichtlich stark unterschiedliche Sichten gibt, die vor einer weiteren Verfeinerung der Anforderungen ausdiskutiert werden sollten.

Das primäre Ziel ist, möglichst viele Agreements, also Gewinnsituationen (Win-Conditions) für alle Beteiligten, zu erhalten, doch auch Risiken und nicht-abgestimmte Win-Conditions müssen für die weitere Planung herangezogen werden.

Eine wichtige Rolle bei EasyWinWin Verhandlungen spielen computergestützte Moderationstechniken. Diese Werkzeugunterstützung reduziert die Komplexität, erleichtert ein strukturiertes Vorgehen und unterstützt die Vollständigkeit im Verhandlungsprozess. Weiter ermöglichen sie die Feststellung und Visualisierung von Einigkeiten und Dissensen bei einer Vielzahl, oft mehreren Hunderten, von Anforderungen. Aufbauend auf diesen Verhandlungen kann eine Priorisierung der Win-Conditions erfolgen. Bei EWW-Workshops werden alle Mitarbeiter des Projektes direkt in das System eingebunden. Die Ergebnisse sind, aufgrund einer automatisierten Protokollierung der Diskussionsprozesse, sofort verfügbar.

Die Hauptvorteile von EasyWinWin sind:

- Verstehen und Abstimmen von unterschiedlichen Erwartungen
- Gemeinsames Erheben und Abklären von Systemanforderungen
- Festlegen von Prioritäten
- Erkennen und Ausräumen von Widersprüchen und Konflikten
- Gemeinsame Definition wichtiger Begriffe

Zur Verdeutlichung der Leistungsfähigkeit von EasyWinWin: die Kategorisierung von 50 bis 120 Win-Situationen benötigt mit Hilfe dieses Groupware-Prozesses nur zwei Stunden, wofür in herkömmlichen Besprechungen früher normalerweise mehrere Tage benötigt wurden.

12.3 Frühe Beurteilung der Qualität: Reviews und Inspektionen

Dieser Abschnitt gibt einen Überblick über einige wichtige Methoden zur Produkt-Qualitätsverbesserung, mit speziellem Augenmerk auf die frühen Schritte im Entwicklungsprozess. Die Hauptgründe dafür, warum gerade in der frühen Entwicklung, also in der Anforderungsanalyse, Planung und Spezifikation, Fehler vermieden werden sollten, liegen darin, dass diese Fehler viel Zeit und Geld kosten können. Das deswegen, weil solche Fehler, wenn sie erst in späteren Phasen des Entwicklungszyklus entdeckt werden (im schlechtesten Fall bei der Übernahme durch den Kunden), sich durch alle vorherigen Phasen ziehen können und damit den Nachbesserungsaufwand oft deutlich vergrößern.

12.3.1 Reviews und Inspektionen

Reviews und Inspektionen sind Methoden der Qualitätssicherung, die während des gesamten Projektes zum Einsatz kommen können. Sie sind nicht auf eine bestimmte Phase im Entwicklungsprozess beschränkt. Genauso wenig sind sie nur auf die Softwareentwicklung beschränkt, im Gegenteil, Reviews und Inspektionen haben sich bereits in nahezu allen technischen Prozessen bewährt.

Die grundsätzlichen Abläufe von Reviews und Inspektionen wurden bereits im Kapitel 5 ausführlich beschrieben. Zusammengefasst sind bei der Durchführung von Reviews und Inspektionen die wesentlichen Schritte *Planung*, *Vorbereitung*, *Umsetzung* und *Korrektur* bzw. *Nachbereitung* notwendig. Unabhängig von der Art des Dokumentes, des Anlasses und der detaillierten Vorgangsweise muss das zu untersuchenden Objekt in geeigneter Form *gelesen* werden. Dieses Kapitel widmet sich den Methoden und Lesetechniken zur konkreten Durchführung der Inspektion bzw. des Reviews.

Checklistenbasierte Lesetechniken: Die Inspektoren, also die lesenden Teammitglieder, untersuchen den Prüfling bezüglich vorgegebener Kriterien. Diese Kriterien, die jeweils auf die Anwendungsdomäne und die konkrete Dokumentart abgestimmt werden müssen, werden als Richtwerte für ein gutes Dokument herangezogen. Die Inspektoren lesen das Dokument im Hinblick auf diese Punkte durch, notieren sich Abweichungen und kommentieren diese. Die Prüfung des Artefakts ist beendet, wenn alle Checklistenpunkte bearbeitet wurden, keine Fehler mehr gefunden werden oder der vordefinierte Zeitrahmen erreicht wird. Checklisten sind besonders gut bei gut bekannten Anwendungsdomänen oder bei ähnlichen Reviewobjekten anwendbar, da bereits auf Erfahrungen zurückgegriffen und ev. existierende Checklisten zurückgegriffen werden kann.

Szenariobasierte Lesetechniken: Dieser Lesetechnikansatz basiert auf definierten Rollen innerhalb des Entwicklungs- bzw. Inspektionsteams. Jeder Inspektor versetzt sich in die Lage einer definierten Rolle, beispielsweise Anwender, Tester, Entwickler und bearbeitet das zu untersuchende Objekt aus dieser Sichtweise. In der Regel gibt es für die jeweiligen Rollen geeignete Leitfäden (*Task-Descriptions*), in denen die wesentlichen Schritte definiert sind. Aufgrund der rollenspezifischen Vorgangsweise müssen diese Leitfäden bei geänderten Anwendungsdomänen nur geringfügig modifiziert werden. Aufgrund der verschiedenen Sichten der Rollen werden unterschiedliche Fehlerarten gefunden. Bei der Anwendung von szenariobasierten Lesetechniken ist zu beachten, dass alle Fehlerkategorien durch die Rollen gefunden werden können.

Detaillierte Informationen und weitere Lesetechnikansätze werden im folgenden Kapitel beschrieben.

Der Zweck einer Inspektion oder eines Reviews ist das Finden von Fehlern. Nach Beendigung einer Review- oder Inspektionssitzung existiert von jedem Teilnehmer eine individuelle Fehlerliste, d.h. eine Liste, in der alle gefundenen Fehler beschrieben sind. Der nächste Schritt umfasst die Zusammenführung aller individuellen Fehlerlisten zu einer so genannten Teamfehlerliste.

Meeting mit Fehlerschätzung: Eine Möglichkeit eine Teamfehlerliste zu produzieren ist ein Teammeeting, in dem alle gefundenen Fehler besprochen und bewertet werden. Bei einem Teammeeting muss darauf geachtet werden, dass alle individuell gefundenen Fehler auch entsprechend behandelt werden. Dazu ist die Rolle eines Moderators unbedingt notwendig. Ein Teammeeting bietet außerdem die Möglichkeit, Unklarheiten zu bereinigen, vermeintliche Fehler zu eliminieren und auch neue Fehler, die "übersehen" wurden, zu finden. Ein weiterer Vorteil eines Meetings ist auch die so genannte Restfehlerschätzung, also die Schätzung der noch nicht gefundenen Fehler. Durch Diskussion innerhalb des Inspektionsteams und der Erfahrung des eigenen Fehlerfindungsprozesses kann diese Schätzung als zuverlässige Größe betrachtet werden.

Die Ergebnisse der Analyse der Fehlerlisten und der Restfehlerschätzungen können für Entscheidungsprozesse herangezogen werden. Das kann sowohl die Freigabe für den weiteren Projektfortschritt (beispielsweise für weitere Entwicklungsphasen) als auch die Wiederholung des Review- oder Inspektionsprozesses oder die Wiederholung der gesamten Phase bedeuten. Erkenntnisse aus den Reviewprozessen werden üblicherweise auch für die Testplanung und weitere qualitätsrelevante Maßnahmen und Kennzahlen verwendet.

12.3.2 Methoden zur Unterstützung von Reviews und Inspektionen

Reviews und Inspektionen zielen primär darauf ab, Fehler bereits in frühen Phasen der Softwareentwicklung zu finden. Dieser Abschnitt stellt einige Ansätze zur Fehlersuche in Softwaredokumenten dar.

Um die Effizienz eines Inspektionsteams zu verbessern, müssen Methoden verwendet werden, die Inspektionsteilnehmern den Umgang mit Dokumenten und das Finden von Fehlern erleichtern. Lesetechniken, d.h. Verfahrensanweisungen zum Lesen eines Softwaredokuments, helfen beim effizienten Finden von Fehlern. Eine Lesetechnik ist eine Reihe von Schritten, die als Anleitung für einen Inspektor dient, einen besseren Zugang und tieferes Verständnis für die betrachtete Softwareentität zu finden [Laitenberger et al, 1999]. Es ist jedoch die Aufgabe des Inspektors, Nutzen aus den Vorteilen einer Lesetechnik zu ziehen.

Die einfachste aller Lesetechniken, die ad-hoc Methode, bei der unsystematisch einfach nach Fehlern gesucht wird, wird zwar in der Praxis immer weniger eingesetzt, liefert aber Vergleichswerte bei der Untersuchung der Effektivität und Effizienz verschiedener Lesetechniken. Die ad-hoc Methode und die checklistenbasierte Lesetechnik werden zu den nicht-systematischen Lesetechniken zusammengefasst. Im Gegensatz dazu stehen die szenariobasierten oder systematischen Lesetechniken. Letztere heißen deswegen szenariobasiert, weil der Inspektor bei der individuellen Fehlersuche jeweils ein Szenario mit den dazugehörigen Aktivitäten durchgeht [Laitenberger et al, 1999], wodurch man gezielt nach Fehlerklassen suchen kann.

Es folgt ein Überblick über die häufigsten der Lesetechniken, es werden jedoch laufend neue Lesetechniken entwickelt, bzw. die Vorteile von mehreren Lesetechniken zu einer neuen kombiniert.

Lesetechnik	**Anwendungsbereich**	**Vorteile**	**Nachteile**
Checklistenbasiert	Traditionelle Methode zur Fehlersuche; derzeit meist „state of the practice“ in Unternehmen	Wenig komplex; viel Literatur und Erkenntnisse verfügbar (Vorschläge für Checklisten); im Aallgemeinen besser als ad-hoc Fehlersuche;	Fehlende Struktur problematisch bei unerfahrenen Reviewern; Für jedes Projekt auf einem neuen Gebiet muss eine komplett neue Checkliste erstellt werden
Fehlerbasiert	Suche nach spezifischen Fehlerklassen. Versuch der Erweiterung und Verbesserung von Checklistenbasierter Lesetechnik	Beschriebene Vorgangsweise für die Reviewer; ansonsten ähnlich Checklistenbasierter Fehlersuche; Vermutlich besser geeignet für unerfahrene Reviewer	Höhere Verantwortlichkeit der Reviewer als bei Checklisten basierten Methoden
Perspektivenbasiert	Wird heute in großen Unternehmen und großen Projekten verwendet, z.B. der NASA;	Gute Fehlerabdeckung, durch die unterschiedlichen Perspektiven; geringe Überschneidung bei der Fehlersuche; je nach Anwendungsbereich kann eine höhere Fehlerfindungsquote als bei Checklistenbasiertem Suchen erzielt werden; Viel Literatur und Experimente vorhanden; Fehlerfindungskosten oft geringer als bei Checklisten;	Teilweise komplizierter in Vorbereitung und Durchführung als Checklistenbasierte Suche
Verwendungsbasiert	Relativ neue Methode; Verwendung für stark kundenorientierte Projekte	Hohe Kundenorientierung; Konzentration auf schwere Fehler bzw. Vernachlässigung von leichteren Fehlern ermöglicht statt „optimaler“ Qualität eine „ausreichende“, oder den Kunden zufriedenstellende, Qualität	Vernachlässigung der Implementierung durch die hohe Kundenorientierung; viele leichte Fehler bleiben unter Umständen unberücksichtigt, die die Verwendbarkeit des Endproduktes herabsetzten

Lesetechnik	Anwendungsbereich	Vorteile	Nachteile
Sample-driven	Reale Verbesserung von Reviewprozessen in Unternehmen	Besserer und budgetorientierter Einsatz von Ressourcen für Reviews; beste Performanz bei stark fehlerdivergenten Reviewobjekten.	Keine, da es sich nicht um eine Lesetechnik handelt, sondern eher um eine Optimierung von deren Einsatz

Tabelle 12.2: Lesetechniken im Überblick

Checklistenbasiertes Lesen (Checklist-based)

Fehlersuche mit checklistenbasiertem Lesen (*Checklist-Based Reading, CBR*) ist die einfachste Lesetechnik nach der ad-hoc Fehlersuche. Diese Methode verwendet, wie der Name schon sagt, eine Checkliste. Anhand dieser Checkliste gehen die Reviewer schrittweise vor und bearbeiten jeden Punkt bzw. jede Frage auf der Checkliste. Sinnvollerweise werden die erledigten Punkte abgehakt und die gefundenen Fehler notiert. Ein Eintrag auf der Checkliste beinhaltet jeweils ein Thema, nach dem das zu reviewende Objekt oder Dokument durchsucht werden soll. Dadurch soll erreicht werden, dass sich der Reviewer immer auf einen eingeschränkten Fehlerbereich konzentriert. Außerdem unterstützt eine Checkliste den Reviewer insofern, dass er immer weiß, worauf er achten soll (im Gegensatz zum ad-hoc Suchen). Checklisten haben den Nachteil, dass sie zum Typ eines Reviewobjekts passen müssen; sie müssen für eine neue Art von Dokument also entweder neu oder anhand von Checklisten aus vergangenen Projekten entwickelt werden. Bei der Wiederverwendung von Checklisten läuft man Gefahr, dass man Fehler die noch nicht aufgetaucht sind, wahrscheinlich wieder nicht findet. Unter Umständen können daher sogar ganze Fehlerklassen unbemerkt bleiben.

Anwenden lässt sich die checklistenbasierte Lesetechnik besonders gut auf ähnliche Reviewobjekte, jedes, von seiner Art, neue Reviewobjekt (z.B. in objektorientierten Design Dokumenten [Laitenberger et al, 1999]) bringt insofern Schwierigkeiten mit sich, dass noch keine Erfahrungen darüber existieren, auf welche Punkte in der Checkliste wert gelegt werden muss.

Die fehlende Struktur bei der Vorgehensweise ist oft ein Problem für unerfahrene Reviewer, so weiß man zum Beispiel nicht, zu welchem Zeitpunkt man genügend Information besitzt um den entsprechenden Checklisteneintrag zufriedenstellend beantworten zu können [Laitenberger et al, 1999]. Auch ist durch die fehlende Vorgehensweise ein späteres Nachvollziehen des Fehlersuchprozesses sehr schwierig. Der Fehlersuchprozess hängt außerdem sehr stark von den individuellen Fähigkeiten des Reviewers ab.

Ein weiteres Problem stellt die Länge der Checkliste dar, da oft zu lange und damit unübersichtliche Checklisten erstellt werden. Gilb und Graham empfehlen als Ergebnis ihrer Forschungsarbeit für die Länge einer Checkliste maximal eine A4 Seite, um die Übersichtlichkeit zu gewährleisten [Gilb et al, 1993].

Lesetechniken unterstützen den Inspektor, indem sie die Form von Fragen definieren, die beim Lesen eines Dokuments bei einer Inspektion beantwort werden müssen [Laitenberger et al, 1999]. Obwohl checklistenbasiertes Lesen bessere Resultate bei der Suche nach Mängeln liefert, benennt Laitenberger et al. vier prinzipielle Schwächen:

- Checklisten repräsentieren die Vergangenheit: Checklisten werden vor Beginn des Inspektionsprozesses erstellt und basieren deshalb auf früheren fehlerhaften Informationen (im schlimmsten Fall passt die Information nicht in den Anwendungsbereich oder es mangelt an Fehlerklassen, falls diese in der Vergangenheit noch nicht aufgetreten sind).
- Ausmaß des Fragenkatalogs: Die Struktur der Lesetechnik führt dazu, dass ein Fragenkatalog ausgesprochen viele Fragen enthalten kann. Es ist für einen Inspektor ziemlich schwierig zu entscheiden, wie eine spezifische Frage zu beantworten ist (es besteht ein Verhältnis zwischen Fragenkatalog und den gefundenen Fehlern).
- Dokumentationsproblem: Wenn ein Inspektor einen Fehler gefunden und im Fragenkatalog verzeichnet hat, scheint keine Notwendigkeit mehr zu bestehen, den Fehler außerhalb des Fragenkatalogs zu dokumentieren. Dies führt zu Mängeln in der Dokumentation oder zwingt andere Inspektoren, den Fehler zu rekonstruieren.
- Berichterstattung: Der Fragenkatalog deckt alle Teile des Dokuments ab, sodass ein Überschuss an unnötigen Details entsteht.

Fehlerbasierte Lesetechnik

Fehlerbasierte Suche ist eine systematische Lesetechnik, mit höherer Verantwortlichkeit der Reviewer [Porter et al, 1995]. Diese Technik wurde ursprünglich entwickelt um Fehler in Anforderungsdokumenten aufzudecken. Für diese Technik werden Fehler in Gruppen aufgeteilt und für jede Fehlergruppe werden genaue Fragestellungen und, im Unterschied zur checklisten basierten Fehlersuche, Vorgangsweisen bei der Fehlersuche entwickelt. Die Reviewer beantworten die Fragestellungen dann indem sie das vordefinierte Szenario durchgehen. Mit einem Szenario kann man meist nicht alle Fehlerklasse finden, deswegen sind mehrere Sichtweisen oder Szenarien oft notwendig.

Die Fehlerfindungsrate ist oft besser als in Checklisten basierter Fehlersuche und besser als bei der ad-hoc Fehlersuche. Bei statistischen Auswertungen von Versuchsergebnissen hat sich gezeigt, dass nicht in allen Fällen eine signifikant bessere Fehlerfindungsrate als bei checklistenbasierter Fehlersuche erzielt werden konnte.

Perspektiven- und Szenariobasierte Lesetechniken

Ursprünglich wurde diese Technik von der NASA, unter entscheidender Mitarbeit von Basili, entwickelt und durch Tests validiert. Diese Lesetechnik basiert auf den unterschiedlichen Interessen und Sichtweisen aller Beteiligten an dem / auf das Reviewobjekt. Jede Perspektive beinhaltet mehrere Fragen und Anforderungen an das Review Objekt. Prinzipiell ergeben sich in der perspektivenbasierten Lesetechnik bei der Vorgehensweise 3 Schritte:

- Erklären der jeweiligen Interessen an dem Reviewobjekt (jeweils aus Kunden-, Designer-, oder Testersicht)
- Spezielle Anweisungen für den Reviewer, die er bei der Fehlersuche berücksichtigen muß
- Eine Liste von Fragen, die während der Fehlersuche beantwortet werden müssen

Man erwartet sich von der perspektivenbasierten Lesetechnik, dass durch die unterschiedlichen Perspektiven keine großen Überlappungen beim Bereich der Fehlersuche stattfinden, d.h. dass wenige Fehler doppelt gefunden werden. Probleme, dass gewisse Fehlerklassen von keiner Perspektive abgedeckt werden, sollen durch diese Lesetechnik ebenfalls verhindert werden. Aus der Tatsache, dass wenige Fehler mehrfach gefunden werden, und die nicht-abgedeckten Fehlerbereiche gering sind, erkennt man eine Kosteneffektivität dieser Lesetechnik.

Als unterschiedliche Rollen für die Perspektiven werden von Basili et al. Kunden, Designer und Tester (in Bezug auf Software) vorgeschlagen [Basili et al, 1996]. Zusätzlich findet man auch öfters die Rolle des Entwicklers in der Literatur.

Für jede der drei Rollen existieren Grundannahmen, was den Zugang zu Testobjekten angeht, so orientieren sich Kunden zum Beispiel an der Vollständigkeit von Anwendungsfällen oder Tester an der Testbarkeit von Teilen des Reviewobjektes.

Der perspektivenbasierte Ansatz gehört zu den meist erforschten und getesteten Lesetechniken, die sich speziell mit Anforderungsdokumenten beschäftigen. Zusätzlich existieren aber noch mehrere Anwendungs- und somit Testbereiche dieser Methode (Design Inspektionen, Code Inspektionen, Usability Tests). Verglichen zur ad-hoc Lesetechnik liefert perspektivenbasiertes Lesen in 3 von 4 Experimenten bessere Ergebnisse. Verglichen zur checklistenbasierten Methode oder fehlerbasierten Methode zeigt sich im perspektivenbasierten Lesen nicht in allen Bereichen immer eine signifikant bessere Fehlerfindungsquote, jedoch werden in einigen Bereichen wie zum Beispiel bei der Inspektion von Designdokumenten bessere Ergebnisse erzielt.

Nach den Ergebnissen von Laitenberger et al. wurden in UML Design Dokumenten eindeutig eine höhere Fehlerfindungsquote nachgewiesen [Laitenberger et al, 1999].

Allgemein scheint es sich herauszukristallisieren, dass die Fehlerfindungskosten bei perspektivenbasiertem Fehlersuchen niedriger sind, als beispielsweise bei checklistenbasierter Suche.

Verwendungsbasiertes (Usage-based) Lesen

Die meisten anderen Lesetechniken zielen darauf ab, möglichst viele Fehler zu finden, unabhängig von ihrer Wichtigkeit. Bei verwendungsbasiertem Lesen (*Usage-Based Reading, UBR*) konzentriert sich der Inspektor auf die schweren Fehler, die das Endprodukt des Reviewobjektes am stärksten beeinflussen. Besser gesagt auf die Fehler, die das Endprodukt aus Benutzersicht am stärksten negativ beeinflussen. Zu diesem Zweck bestimmt man Anwendungsfälle und ordnet diesen Prioritäten zu. Am Schluss sollte man eine, nach Prioritäten geordnete, Liste erhalten, nach der man vorgehen kann. Der Anwendungsfall mit der höchsten Priorität stellt den kritischsten Anwendungsfall dar, daher sollte bei diesem am wenigsten ein Fehler auftreten. Der Reviewer startet nun mit dem Anwendungsfall mit der höchsten Priorität, und geht die Anwendungsfälle der Reihe nach durch. Für den Fall, dass nicht alle Anwendungsfälle bearbeitet werden können, möglicherweise aus Zeitmangel oder ähnlichem, wird einfach an einer Stelle in der Liste abgebrochen.

Es besteht eine gewisse Ähnlichkeit zur Kundensicht im PBR (perspective-based-reading), da diese im Normalfall auch mit Anwendungsfällen arbeitet. Meistens werden jedoch im Zuge der Fehlersuche im PBR die Anwendungsfälle erst entwickelt, im UBR sollten sie aber schon vorab definiert sein. Im Unterschied zu PBR ist UBR auch nicht für unterschiedliche Szenarios der glei-

chen Kategorie (z.B. Anforderungsdokumente) zu verwenden. Das Szenario muss in UBR für jedes Projekt neu entwickelt werden, weil die Anwendungsfälle ja schon vorher feststehen.

Da die Idee von UBR relativ neu ist gibt es noch nicht viele Erfahrungen auf diesem Gebiet. UBR ist aber nach einem Experiment von Thelin durchaus erfolgversprechend [Tehlin, 2002]. Bei einem Vergleich mit CBR ergab sich eine signifikant höhere Fehlerfindungsrate bei kritischen Fehlern, allerdings nicht bei allen Fehlern. Es werden also eher kritische Fehler gefunden. Auch werden Fehler generell früher entdeckt, und zwar gilt das für alle Fehler.

Sample-driven Reviews

Die Motivation hinter dieser Lesetechnik ist der große Druck zur schnellen Markteinführung eines neuen Produkts, bzw. zu oftmaligen Updates eines Produktes. Deswegen sollen Reviews möglichst effizient sein, weil die dort benutzten überflüssigen Ressourcen anderweitig (z.B. in der eigentlichen Produktentwicklung) verwendet werden könnten. Leider wird in vielen Unternehmen die Produktivität in Codezeilen (Lines of Code, LOC) gemessen, und dabei kann ein Review wenig vorweisen. Oft sind Reviewtätigkeiten daher an erster Stelle bei Einsparungsmaßnahmen. Daher versucht man in solchen Fällen sich auf die wichtigsten Objekte zu konzentrieren, da diese Wichtigkeit nicht klar ersichtlich ist, muss sie geschätzt werden. Man konzentriert sich auf eine kleine Auswahl der zu reviewenden Objekte, bzw. auf Teile des zu reviewenden Objektes (daher auch Sample-Based) und versucht jene zu identifizieren, deren Qualität am schlechtesten ist. Es gibt mehrere Sampling-Strategien, von denen nicht jede bei jedem Reviewobjekt sinnvoll ist. Dieses Sample von Reviewobjekten wird nun auf Fehler untersucht und nach Abschluss der Untersuchung wird anhand des Samples auf die Qualität aller Objekte geschlossen. Dies dient als Entscheidungshilfe für die Ressourcenzuteilung für die nachfolgende, eigentliche Reviewphase.

Sample-basierte Fehlersuche ist also keine eigene Fehlersuchmethode sondern eher eine mögliche Erweiterung für den praktischen Gebrauch von Lesetechniken!

Basierend auf den Erkenntnissen von Thelin et al. ist die sample-driven Methode besser als zufällig ausgewählte Exemplare (dies bezieht sich auf die Verwendung von Sampling Methoden zur Zusammenstellung eines Samples). Die Methode hat sich als zuverlässig herausgestellt, vor Allem bei Reviewobjekten mit einer hohen Differenz in der Fehlerdichte. Der Vorschlag bei einer normalen Größe eines Reviewobjektes, ist 20-30% des Review Objektes als Sample von einem Reviewer durchsuchen zu lassen um die gewünschten Informationen über das Review Objekt zu erhalten.

Softwareinspektion unter Zuhilfenahme bestimmter Lesetechniken ist ein nützliches Instrument um Fehler in Anforderungsdokumenten aufzuspüren. Weil eine große Menge unterschiedlicher Lesetechniken existiert ist es interessant herauszufinden, welcher Ansatz am besten geeignet ist um so viele Fehler wie möglich zu finden. Weil Inspektionen Teamaktivitäten sind, kann es sich als nützlich erweisen, verschiedene Lesetechniken innerhalb des Teams zu kombinieren, um bessere Ergebnisse zu erzielen.

12.4 Assessments / Audits

Reviews, Inspektionen oder Tests werden üblicherweise auf konkrete Produkte und Teilprodukte angewandt und laufen nach einem definierten Prozess ab. Die Verbesserung bzw. Überprüfung der

eigentlichen Prozesse läuft im Normalfall innerhalb eines Regelkreises ab, wie er beispielsweise im PDCA-Zyklus bereits beschrieben wurde.

Audits und Assessments zielen grundsätzlich auch in die Richtung, ein Produkt, einen Prozess oder ein System zu überprüfen, folgen allerdings einem formaleren und abstrakterem Ablauf. Beispielsweise wird durch Audits und Assessments die Konformität eines Prozesses zu einem Standard (z.B. QM-Standard oder SE-Standard) mit dem Ziel, Abweichungen festzustellen, Verbesserungspotential zu finden oder beispielsweise ein Zertifikat zu erhalten, geprüft. Die Ergebnisse werden als Eingabegröße für die Verbesserung des untersuchten Prozesses verwendet.

Qualitätsmodelle, wie beispielsweise das EFQM-Modell (siehe nächstes Kapitel) betrachten den kontinuierlichen Verbesserungsprozess von Produkten, Prozessen und Managementsystemen als Grundvoraussetzung für ein erfolgreiches Unternehmen. Durch die fortlaufende Anwendung von Audits und Assessments werden diese Anforderungen überprüft und helfen, auf Unternehmensebene effektive und effiziente Entwicklungsprozesse herzustellen und zu optimieren. Gut abgestimmte Entwicklungsprozesse unterstützen wiederum eine qualitativ hochwertige Softwareherstellung und schaffen wesentliche Voraussetzungen für erfolgreiches Testen.

Diese formal gehaltenen Überprüfungen dienen einerseits der Feststellung der Konformität der unternehmenseigenen Vorgaben mit den Vorgaben der einschlägigen Normen und andererseits dem Vergleich zwischen IST und SOLL innerhalb eines Unternehmens.

Allgemein wird ein Assessment bzw. ein Audit als unabhängige Überprüfung von Produkten, Prozessen, Managementsystemen, etc verstanden. Das American National Standards Institute (ANSI) definiert eine derartige Überprüfung folgendermaßen:

> *"An activity to determine through investigation the adequacy of, and adherence to, established procedures, instructions, specifications, codes and standards or other applicable contractual and licensing requirements, and the effectiveness of implementation" [ANSI N45.2.10-1973].*

	Audit (DIN EN ISO 9001)	**CMM**
Basis	Standard, Norm	Beschreibung der Schlüsselprozesse
Abdeckung	Standard	Software Norm / Standard
Methode	Audit	Assessment
Prüfer	2-4 Personen; Nicht ausschließlich Softwarespezialisten	3-5 Personen; Ausschließlich Software Spezialisten
Dauer	1-2 Tage	Circa 1 Woche
Ergebnis	Rating	Zertifikat
Ziele	Zertifikat, falls ein Unternehmen die Anforderungen der entsprechenden ISO 9000x Reihe erfüllt. Keine Verbesserungsvorschläge	Klassifizierung eines Unternehmens (z.B. Reifegrad gemäss CMM-Modell) Analysieren und verstehen der benutzten Prozesse und Techniken

	Audit (DIN EN ISO 9001)	CMM
	Stellt die Qualitätsfähigkeit des Unternehmens, seiner Produkte und Prozesse dar.	Finden von Stärken und Schwächen Ermittlung des Verbesserungspotentials Beschreibung der (Sub-) Prozesse für weitere Verbesserungen

Tabelle 12.3: Vergleich ISO vs. CMM und Audits vs. Assessments; nach [Thaller, 2000]

Thaller beschreibt Assessments als Rating oder Review einer Organisation und der Prozesse der Softwareentwicklung [Thaller, 2000].

Bei einem Zertifizierungsaudit nach ISO 9000wird das unternehmenseigene QM-System auf der Basis des QM-Handbuchs, das in Übereinstimmung mit den jeweiligen ISO Standards erstellt wurde, einerseits gegen die Normenvorgabe anderseits gegen die betriebliche Praxis geprüft. Assessments werden vom Management oder einer dritten Partei, so genannten Zertifizierungsstellen (*certification authorities*), im Falle einer Zertifizierung, durchgeführt, um beispielsweise bei CMM einen definierten Reifegrad nachweisen zu können.

Beide Methoden, Audits und Assessment zielen darauf ab, Problem- und Schwachstellen zu finden, Informationen über bestimmte Themen einzuholen oder diese Themen gegen bestimmte Anforderungen, Standards oder Spezifikationen zu überprüfen. Produkte, Prozesse oder das gesamte QM-System kann Thema eines Audits bzw. Assessment sein. Tabelle 12.4 1 zeigt einige Unterschiede zwischen Assessments und Audits im Überblick

Audits werden in vielen verschiedenen Unternehmensbereichen oder Projekten eingesetzt, um Schwachstellen aufzudecken und Verbesserungspotentiale zu finden. In speziellen Auditberichten werden diese Ergebnisse protokolliert und als Basis für die Ermittlung von Verbesserungsmaßnahmen verwendet.

Pfeiffer unterscheidet verschiedene Arten von Audits mit anwendungsspezifischen, speziellen Vorgehensweisen [Pfeiffer, 1996]:

- *Produktaudit*: Im Vordergrund steht das (Teil-)Produkt, das auf formale Übereinstimmung mit den Vorgaben geprüft wird.
- *Prozessaudit*: Definierte Prozesse innerhalb eines Unternehmens sind auf verschiedenste Arten dokumentiert (Prozessbeschreibungen, QM-Handbuch, usw.). Prozessaudits überprüfen die tatsächliche Vorgangsweise (Praxis) im Hinblick auf die Soll-Abläufe (Theorie).
- *Systemaudit*: Auf höchster Abstraktionsebene wird das Managmentsystem (QM-System) gegen die definierten Vorgaben und Standards bzw. Normen geprüft.

Produktaudits fokussieren auf ein fertiges Teil- oder Endprodukt oder – im Softwareengineering Kontext – Ergebnisse einer bestimmten Phase innerhalb des Softwarelebenszyklus. Die Hauptziele beider Ansätze sind es, die Qualität zu prüfen, Defekte und deren Ursachen zu finden und letztlich Verbesserungsvorschläge zu entwickeln [Pfeiffer, 1996]. Während Produktaudits die Produkte an

sich betreffen, werden mit Prozessaudits Abläufe (Workflows) und die benutzten Methoden, mit dem Fokus auf Optimierung, überprüft und analysiert.

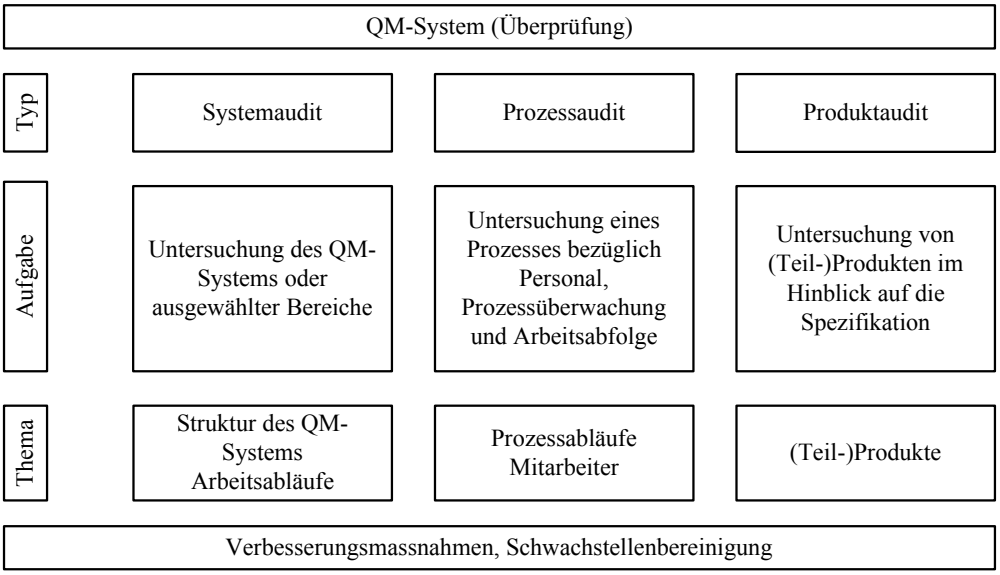
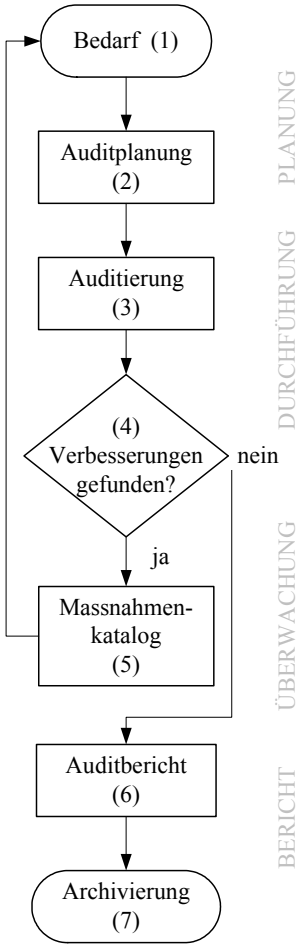


Abbildung 12.3: Audittypen

Systemaudits werden dazu benutzt, um ein gesamtes QM-System oder kritische Teilbereiche auf Korrektheit und Effizienz zu prüfen. Wenn die Anforderungen nicht erfüllt werden oder sogar Fehler gefunden werden, müssen Maßnahmen zur Korrektur getroffen werden. Die Basis für ein Systemaudit ist beispielsweise ein QM-Handbuch, die Art der Überprüfung kann als Assessment für eine Zertifikation verstanden werden.

Ein weiteres Unterscheidungsmerkmal erfolgt zwischen internen und externen Audits. Während Kunden oder Zertifizierungsstellen externe Audits durchführen, um die Korrektheit des QM Systems zu überprüfen, liegt der Schwerpunkt von internen Audits auf der Optimierung der Produkte, Prozesse innerhalb des Unternehmens. Interne Audits können einerseits als effektive Managementmethode und andererseits als geeignetes Werkzeug zur Feststellung der eigenen Qualitätsfähigkeit bzw. zur Verbesserung der Arbeitsabläufe verstanden werden. Abbildung 12.3 zeigt die verschiedenen Ausprägungen von Audits im Vergleich.

Tabelle 12.4 zeigt das typische Vorgehen bei der Durchführung eines Audits, das beispielsweise in einem QM Handbuch nach ISO 9001 im Kapitel "Interne Qualitätsaudits" dokumentiert wird.



Pos	Tätigkeit
1	*Bedarf eines Audits* • Definierter Meilenstein innerhalb eines Phasenplans • Geplantes Audit gemäss Projektplan • Kundenforderung • Routineaudits durch Projektmanagement oder QS
2	*Planung eines Audits* • Themenbereich (welcher Bereich muss auditiert werden) • Festlegung des Auditteams und der Kontaktpersonen • Formale Basis festlegen (Norm, Checklisten, usw.) • Koordination (Zeitplan, Personen, usw.)
3	*Durchführung eines Audits* • Auditdurchführung gemäss Auditplan • Dokumentation der Ergebnisse • Diskussion der Ergebnisse mit dem Teilnehmerkreis
4	*Verbesserungspotential / Schwachstellen* • Diskussion von Stärken, Schwächen und Verbesserungspotential • Festlegung eines Wiederholaudits (falls notwendig) • Auditbericht
5	*Einleitung von Verbesserungsmassnahmen* Umsetzung des Verbesserungspotentials (entweder zur Beseitigung von Schwachstellen oder im Rahmen des ständigen Verbesserungsprozesses)
6	*Auditbericht* Erstellung eines Berichts mit Auditergebnissen und abgestimmten Verbesserungsmassnahmen
7	*Archivierung und Abschluss*

Tabelle 12.4: Planung und Durchführung von Audits

Audits und Assessments werden hauptsächlich auf Management Ebene, allerdings auch während des gesamten Lebenszyklus (z.B. Prozess- und Produktaudit), untersucht und beurteilt. Wenn eine Software einmal geschrieben wurde, muss sie auch getestet werden, ob sie die Funktionalität garantiert. Exemplarisch werden an dieser Stelle Reviews genannt.

12.4.1 European Foundation for Quality Management (EFQM)

Durch die immer größer werdenden wirtschaftlichen Anforderungen an Unternehmen, ist es sinnvoll, ein universal einsetzbares Messinstrument für die Bewertung des Managements eines Unternehmens zu verwenden. Dadurch können wichtige Kriterien der Organisation sowie Geschäftsergebnisse einer Firma gemessen werden, um daraus Rückschlüsse zur Verbesserung zu ziehen.

> *"Man braucht etwas, was einem sagt 'Wenn Du das tust, wirst Du langfristig Erfolg haben'. Und genau das tut dasModell." [Sir Peter Bonfield, CEO von BT]*

Auf diesen Ideen aufbauend wurde eine europäische Non-Profit Organisation, bestehend aus 800 Firmen, gegründet, die mit Hilfe eines aus 9 Kriterien bestehenden Modells ein optimales Managementvorgehen für alle Mitglieder dieser Organisation definieren kann.

Überblick

Die "European Foundation for Quality Management" wurde 1988 von 14 großen europäischen Konzernen unter der Schirmherrschaft der Europäischen Kommission gegründet. Diese Unternehmen sind Bosch, BT, Bull, Ciba-Geigy, Dassault, Electrolux, Fiat, KLM, Nestlé, Olivetti, Philips, Renault, Sulzer und Volkswagen. Die Organisation ist eine Non-Profit Organisation und besteht derzeit aus fast 800 Mitgliedern – Firmen und Organisationen.

Sie benutzt das sogenannte "EFQM-Excellence Model" - siehe Abbildung 12.4 - als strategisches Hilfsmittel für ihr Management. Dieses beruht auf den Grundsätzen des Total Quality Management (TQM), stellt einen integrierten Ansatz zum Qualitätsmanagement dar und umfasst alle Schlüsselfaktoren wie Geschäftsergebnisse, Kundenzufriedenheit, Führung etc. Das EFQM Modell wurde im Jahr 1991 herausgegeben.

Das EFQM-Modell, eine aus neun Kriterien bestehende, offen gehaltene Grundstruktur, kann zur Bewertung des Fortschritts einer Organisation in Richtung Exzellenz herangezogen werden. Das Modell berücksichtigt die vielen Vorgehensweisen, mit denen nachhaltige Exzellenz in allen Leistungsaspekten erzielt werden kann.

Exzellente Organisationen werden an ihrer Fähigkeit gemessen, überragende Ergebnisse für ihre Interessengruppen zu erwirtschaften und aufrechtzuerhalten. Überragende Ergebnisse zu erwirtschaften ist schon schwierig genug – noch schwieriger aber ist es, sie in einer Welt des immer stärker werdenden globalen Wettbewerbs, rascher technologischer Innovation, sich ständig ändernder Arbeitsprozesse und des häufigen Wechsels im wirtschaftlichen, sozialen und kundenbezogenen Umfeld aufrechtzuerhalten. Die "European Foundation for Quality Management" (EFQM) erkannte diese Herausforderung und wurde gegründet, um ein Managementvorgehen für alle Organisationen in Europa zu entwickeln, das zu nachhaltiger Exzellenz führt. [EFQM, 2003]

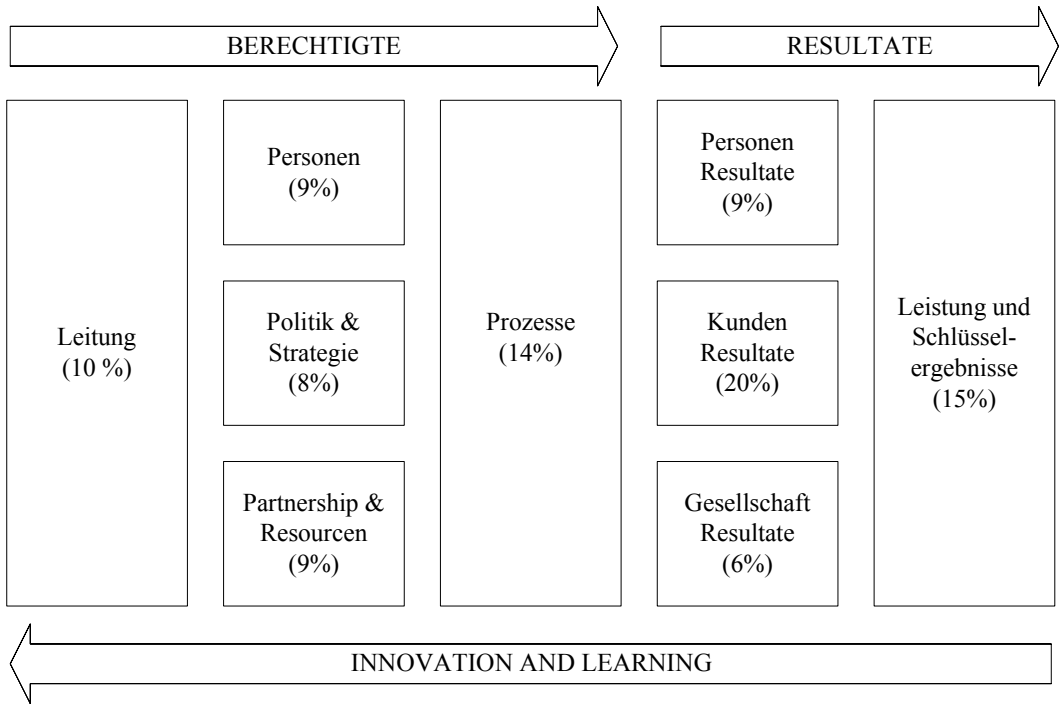


Abbildung 12.4: Berechtigte und Erfolgsfaktoren

Die acht Eckpfeiler des Excellence Modells:

- *Ergebnisorientierung*: Exzellent ist es, Ergebnisse zu erzielen, die alle Beteiligten zufrieden stellen.
- *Kundenorientierung*: Exzellent ist es, nachhaltige Werte für den Kunden zu produzieren.
- *Führung und Zielkonsequenz*: Exzellent ist visionäre und inspirative Führung, gekoppelt mit konsequenter Zielverfolgung.
- *Management mit Prozessen und Fakten*
- *Mitarbeiterentwicklung und -beteiligung*: Exzellent ist es, die Ergebnisse der Mitarbeiter durch deren Anteilnahme zu maximieren.
- *Kontinuierliches Lernen, Innovation und Verbesserung*
- *Aufbau von wertschöpfenden Partnerschaften*
- *Verantwortung gegenüber der Öffentlichkeit*: Ethisch einwandfreies Vorgehen, das die Erwartungen und Regeln der Gesellschaft weitestgehend übertrifft, zur Steigerung der Glaubwürdigkeit, der Leistung und der Wertschätzung der Organisation.

EFQM stellt einen Handlungsrahmen für das Management dar und erlaubt die Einschätzung der Reife eines Unternehmens in Bezug auf seine Markttüchtigkeit durch Selbstbewertung.

Einsatz in der Softwareentwicklung

Das EFQM-Modell ist ein globales Modell und wurde allgemein zur Prozessbewertung entwickelt und nicht speziell für Softwareentwicklungsprozesse. Es erlaubt eine wesentlich abstraktere Sichtweise, die zu einer besseren Gesamtsicht des Unternehmens führt, aber keine detaillierten Analysen des Softwareentwicklungsprozesses zulässt. Es stellt einen Handlungsrahmen für das Management dar und erlaubt die Ermittlung der Reife eines Unternehmens hinsichtlich der Markttüchtigkeit.

Um eine Selbstbewertung nach EFQM durchführen zu können, wird zunächst eine Unternehmensbeschreibung angefertigt, die gegliedert nach den Bewertungskriterien die Situation im Unternehmen möglichst objektiv und vollständig beschreibt. Diese Beschreibung dient anschließend den Bewertungsteams als Grundlage für die Beurteilung. Das EFQM-Modell enthält auch das Bewertungsverfahren, nach dem die Kriterien beurteilt werden. Die Beurteilung erfolgt nach dem Grad der Überdeckung zwischen der Unternehmensbeschreibung und der Kriterienbeschreibung im Modell. Jedes Bewertungsteammitglied bewertet jeden Unterpunkt der Nährbodenkriterien, indem es zwei Faktoren miteinander kombiniert. Der erste Faktor beschreibt, wie gut das Vorgehen ist, der zweite, wie umfangreich das Vorgehen umgesetzt wird. Durch dieses Verfahren erhält man eine Stärken- und Schwächenanalyse, aus der ein entsprechender Aktionsplan erarbeitet werden kann.

Dieser allgemeingültige Ansatz ist für Softwareproduktionsunternehmen ohne Veränderung anwendbar, jedoch nicht spezifisch.

Vorteile / Nachteile

Vorteile

- Integrierter Ansatz
- Starke Fokussierung auf Ergebnis und Kunde
- Liefert brauchbaren Überblick und Verständnis für die Gesamtaspekte des Unternehmens
- Die Kriterien erfassen alle Konzernangelegenheiten

Nachteile

- Komplex und abstrakt
- Nicht speziell auf den Softwareentwicklungsprozess angepasst
- Erfahrung mit dem Modell notwendig
- Keine Vorschläge zur Implementierung
- Abhängig von der Akzeptanz

12.5 Zusammenfassung

Dieses Kapitel behandelte Ansätze zur Beurteilung und Verbesserung der Produkt- und Prozessqualität, die einerseits ein gutes Umfeld für das Testen von Software schaffen und andererseits die Testaktivitäten in der Entwicklung ergänzen können und sollen.

Flexible Ansätze für die Entwicklung. Traditionelle Methoden zur Softwareentwicklung, wie sie derzeit weit verbreitet genutzt werden, zeigen starke Schwächen im Umgang mit sich raschen ändernden Anforderungen. Agile Softwareentwicklung verspricht einen Ansatz zur Lösung dieser Herausforderungen durch schlanke und kundenorientierte Vorgehensweisen. Agil heißt "beweglich, flink, gewandt" und bedeutet im Kontext der Softwareentwicklung, das Projekt flexibel genug zu halten, um die aktuellen Aufgabenstellungen der Auftraggeber zu lösen - und nicht die Probleme, die der Auftraggeber vor zwei Jahren, beim Start des Projekts, hatte. Das "Manifesto for Agile Software Developing", postuliert *Individuen und Interaktionen* wichtiger zu nehmen als Prozesse und Werkzeuge, *Laufende Software* wichtiger als umfangreiche Dokumentation, *Einbeziehung des Kunden* wichtiger als Vertragsverhandlungen, *Auf Änderungen zu reagieren* wichtiger als an Plänen festzuhalten. Diese Punkte liegen allen agilen Methoden zugrunde. In diesem Kapitel wurden die methodischen Ansätze: eXtreme Programming (XP), SCRUM, Familie der Crystal Methoden, Dynamic Systems Development Model (DSDM) und Adaptive Software Development (ASD) behandelt.

Groupware-Systeme zur Unterstützung für Workshops zur Anforderungserfassung. Unabhängig von der Umsetzung und der verwendeten Entwicklungsmethodik ist es notwendig, die Anforderungen an das zu erstellende System zu erkennen. Klare Anforderungen sind die Grundlage für ein erfolgreiches Softwareprojekt, das Bestimmen jener ist jedoch oft ein komplexer und schwieriger Prozess. Bei der Erstellung der Anforderungsdefinition müssen sich nicht nur alle Beteiligten ihre unterschiedlichen Erwartungen einbringen, die Anforderungen müssen auch exakt definiert und – oft in langwierigen und zeitaufwendigen Besprechungen – verhandelt werden. Zur Unterstützung dieses Prozess werden sogenannte Groupware-Systeme, die zur Erleichterung und Verbesserung der Kommunikation verwendet. EasyWinWin ist ein Beispiel für eine mögliche Realisierung dieser Anforderungen. Die aus der Spieltheorie kommende Bezeichnung Win-Win definiert Bedingungen zwischen zwei oder mehreren Beteiligten, bei denen jeder gewinnen kann. So ist es auch in Softwareprojekten als Ziel eine Win-Win Situation anzustreben. EasyWinWin hilft bei der Erhebung und Verhandlung von Anforderungen und der Analyse unterschiedlicher Erwartungen in einem Projektteam. Außerdem ermöglicht es die Identifikation und Visualisierung von Einigkeiten und Dissensen bei oft mehreren hunderten Anforderungen.

Frühzeitige Reviews mit leistungsfähigen Lesetechniken. Nicht nur die Klarstellung und Definition von Anforderungen, auch das Vermeiden von Fehlern in den frühen Phasen eines Projekts ist essentiell, da sich Fehler, die im Design gemacht werden, durch alle anderen Phasen ziehen und deren Korrektur sehr viel Zeit und Geld kostet. Reviews und Inspektionen (im Softwareentwicklungsprozess) können mit den aktuellen Lesetechniken effektiv und effizient Fehler in frühen Phasen der Softwareerstellung finden helfen und damit unproduktive Nachbearbeitungsaufwände vermeiden.

Ansätze zur Prozessbeurteilung und -verbesserung. Audits und Assessments (auf Produkt-, Prozess- und Managementebene) und Qualitätspreise, wie der European Quality Award beschreiben eine strukturelle Vorgangsweise und unterstützen das Unternehmen dabei, gute Produkte und

Dienstleistungen zu erstellen und zu erbringen. Der Einsatz dieser Methoden ist allerdings nicht ausreichend, hoch qualitative Leistungen zu erbringen, sie erleichtern nur die entsprechende Vorgangsweise durch die Bereitstellung geeigneter Rahmenbedingungen. Die "European Foundation for Quality Management", eine Organisation großer europäischer Industrieunternehmen, vergibt seit 1988 den European Quality Award. Das zur Beurteilung von Unternehmen benutzte "EFQM Excellence Model", ein aus neun Kriterien bestehendes Framework, stellt einen Handlungsrahmen für das Management dar. Das EFQM-Modell ist ein globales Modell und wurde allgemein zur Prozessbewertung entwickelt und nicht speziell für Softwareentwicklungsprozesse. Es erlaubt eine wesentlich abstraktere Sichtweise, die zu einer besseren Gesamtsicht des Unternehmens führt.

12.6 Literaturreferenzen

[ASD, 2003] Adaptive Software Development (ASD); Internet: http://www.adaptivesd.com/.

[Agile Alliance, 2003] Agile Alliance; Internet: http://www.agilealliance.com.

[Basili et al, 1996] Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sorumgard, S.and Zelkovitz, M.V., „The Empirical Investigation of Perspektive-Based Reading“, *Empirical Software Engeneering: An International Journal*, 1(2): 133-164, 1996.

[Crystal, 2003] Crystal Familie; Internet: http://crystalmethodologies.org/.

[DSDM, 2003] DSDM; Internet: http://www.dsdm.com/.

[EWW, 2003] Easy-Win-Win (EWW): A Groupware-Supported Methodology For Requirements Negotiation; Internet: http://sunset.usc.edu/research/WINWIN/EasyWinWin/.

[EFQM, 2003] European Foundation for Quality Management (EFQM); Internet: http://www.efqm.org/.

[Gilb et al, 1993] Gilb,T. and Graham, D., *Software Inspections*, Addison-Wesley, ISBN 0-210-63-181-4, 1993.

[Highsmith, 2000] Highsmith, J. A.: “Adaptive Software Development: A Collaborative Approach to Managing Complex Systems”, 2000.

[ISO, 2003] ISO; Internet: http://www.iso.ch/.

[Laitenberger et al, 1999] Laitenberger, Oliver; Atkinson, Colin; Schlich, Maud; El Emam, Khaled: „An Experimental Comparison of Reading Techniques for Defect detection in UML design Documents”, 2000, IESE-Report 080.99/E, December 1999; INSERN, 2000-01.

[Manifest, 2003] Das Agile Manifest; Internet: http://agilemanifesto.org/.

[Polen, 2002] Polen, Michael; McCabe, Rich: “Software Productivity Consortium”, Oct 2002.

[Pfeiffer, 1996] Pfeiffer, Tilo: „Qualitätsmanagement, Strategien, Methoden, Techniken, 2. Auflage“, München, Wien – Hanser; 1996, ISBN 3-446-18579-8.

[Porter et al, 1995] Porter, A.A., Votta, L., Basili, V.R., „Comparing Detection Methods for Software Requirements Inspection: A Replicated Experiment“, in *IEEE Transactions on Software Engeneering*, Vol 21, No. 6, pp. 563-575, 1995.

[Scrum, 2003] Scrum; Internet: http://www.controlchaos.com/.

[Starke, 2002] Starke, Gernot: "Effektive Software Architekturen-Ein praktischer Leitfaden", Hanser Verlag, ISBN 3-446-21998-6, 2002.

[Thaller, 2000] Thaller, Georg Erwin: „Software Qualitat", VDE Verlag, 2000, ISBN: 3-800-72494-4.

[Thelin et al, 2001] Thelin. T., Petersson, H., Wohlin, C., „Sample-driven Inspections", *Proceedings Workshop on Inspection in Software Engineering (WISE'01)*, pp. 81-91, Paris, France, July 2001.

[Thelin, 2002] Thelin, T., „Empirical Evaluation of Usage-Based Reading and Fault Content Estimation for Software Inspections" , PhD Thesis, Sept. 2002.

[Wallmüller, 2001] Wallmüller, Ernest: „Software-Qualitätsmanagement in der Praxis", 2. völlig überarbeitete Auflage, Hanser Fachbuch, 2001, ISBN: 3-446-21367-8.

[XP, 2002] Extreme Programming; A gentle introduction; Internet: http://www.extremeprogramming.org/, http://www.xprogramming.com/.

[Zuser et al, 2001] Zuser, Wolfgang; Biffl, Stefan; Grechenig, Thomas; Köhle, Monika: „*Software Engineering mit UML und dem Unified Process*", Pearson Studium, 2001, ISBN: 3-827-37927-2.

12.7 Übungen und Fragen

1. Geben Sie einen kurzen Überblick über agile Methoden.
2. Auf welchen Ideen baut Extreme Programming auf?
3. Was versteht man unter User Stories?
4. Von welchen beiden Faktoren hängt die Auswahl einer geeigneten Crystal-Methode ab?
5. Verdeutlichen Sie das Prinzip der automatisierten Tests beim Extreme Programming.
6. Was versteht man unter einem Assessment in Bezug auf Qualitätsmanagementsysteme? Worauf zielen Assessments ab?
7. Was sind Kriterien der Ernte beim EFQM-Modell? Was kann durch sie gemessen werden?
8. Anforderungsbeschreibung:

   Sie haben gerade eine international tätige Software-Entwicklungsfirma gegründet, die 9 Mitarbeiter beschäftigt, welche auf 3 Büros in 2 verschiedenen Ländern verstreut sind. Sie haben gerade Ihren ersten größeren Auftrag bekommen, und wollen mit der Anforderungsbeschreibung beginnen. Geben Sie eine mögliche Vorgangsweise an und begründen Sie Ihre Entscheidung.
9. Management von Prüfungen:

Sie sind bei einer Softwarefirma für das Management der Fehlerprüfung verantwortlich. In welchen Phasen werden Sie verstärkt Fehler suchen? Begründen Sie Ihre Antwort.

10. Verbesserung des Testprozesses:

    Sie sind Projektleiter dieses Software-Projektes und haben in Ihrem letzten Projekt festgestellt, dass einige gravierende Fehler bis zum Kunden vorgedrungen sind. Was wollen Sie nun in Ihrem Team verbessern?