Lerneinheit 1

Qualität im Software-Engineering

Die Qualitätssicherung im Software-Engineering strebt nicht nur nach einem möglichst fehlerfreien Endprodukt, sondern zielt vor allem auf eine Qualitätssteigerung durch laufende Verbesserungen im Entwicklungsprozess ab. Diese Lerneinheit zeigt einige der Ansatzpunkte für die Qualitätssicherung.

Lernen

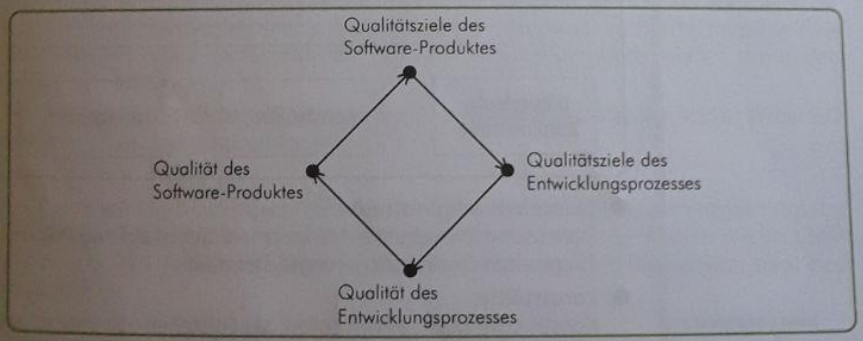
1 Software-Qualität

"You can't test quality into a product."

Die grundlegenden Fragen, mit welchen sich die Software-Qualitätssicherung beschäftigt, sind:

- Definition des Begriffes Software-Qualität
- Verfahren zur Qualitätsprüfung von Software
- Verfahren zur Qualitätslenkung im Software-Entwicklungsprozess

Entscheidend dabei ist, dass Qualitätssicherung nicht etwas ist, das beim (fast) fertigen Produkt ansetzt, sondern bereits bei der Gestaltung des Entwicklungsprozesses. (Das Software-Produkt umfasst dabei die Teile Source Code, Object Code und Dokumentation.)



Ausgehend von den Qualitätszielen des Software-Produktes ist es erforderlich, auch entsprechende Qualitätsziele des Entwicklungsprozesses zu definieren. Kann die geforderte Qualität des Entwicklungsprozesses erreicht werden, so ist auch eine Qualität des Software-Produktes (entsprechend den eingangs formulierten Qualitätszielen) möglich.

Wichtige Begriffe der Qualitätssicherung sind die „Qualität" und das „Merkmal" (nach DIN 55350):

- **Qualität** ist die Gesamtheit von Eigenschaften und Merkmalen eines Produkts oder einer Tätigkeit, die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse bezieht.
- Ein **Merkmal** ist jene Eigenschaft, die eine quantitative oder qualitative Unterscheidung eines Produktes oder einer Tätigkeit aus einer Gesamtheit ermöglicht.

Vgl. Kapitel 7, Lerneinheit 1 (Software-Krise)

Eine wesentliche Motivation zur Qualitätsverbesserung im Software-Entwicklungsprozess ist der „application backlog" – unerledigte Entwicklungsaufträge „stauen" sich gleichsam, sie können nicht bearbeitet werden, da die Entwickler einen Großteil ihrer Zeit mit der (Fehler behebenden) Wartung bestehender Applikationen beschäftigt sind. Die Produktivität der Entwickler kann verbessert werden,

CASE: Computer Aided Software Engineering

- indem automatische (CASE-)Werkzeuge und Tools die Arbeitsintensität verringern bzw.
- durch Produktivitätsverbesserung im konventionellen Entwicklungsprozess, etwa durch Methoden.

Vgl. Kapitel 8, Lerneinheit 3

Eine Anhebung des Produktivitätsniveaus ist nur durch eine gleichzeitige Anhebung von Effizienz und Qualität möglich (vgl. Abbildung auf Seite 264). (Qualität und Effizienz sind bei konstanter Produktivität konkurrierende Ziele.) Die Produktivität kann gesteigert werden durch

Maßnahmen zur Produktivitätssteigerung

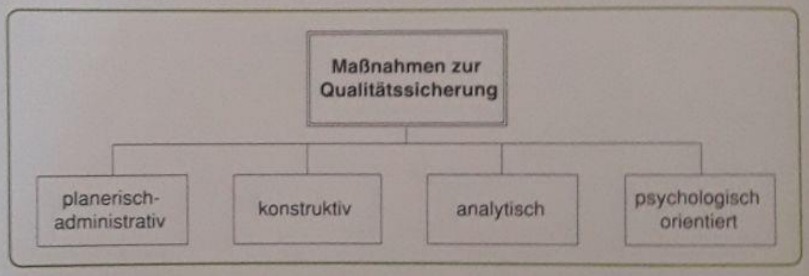
- „Reuse" (Wiederverwendung) von bestehenden Entwürfen und Codes,
- ein verbessertes Vorgehensmodell,
- den Einsatz neuer Software-Entwicklungstools,
- Anwendung der aktuellen Prinzipien des Software-Engineerings.

2 Maßnahmen zur Qualitätssicherung

Qualitätssicherung kann nur dann wirksam werden, wenn sie alle Aspekte des Software-Entwicklungsprozesses berücksichtigt.

Qualitätssicherungsmaßnahmen lassen sich nach folgenden Gesichtspunkten beschreiben:

Maßnahmen zur Qualitätssicherung



- **planerisch-administrativ:**
  Planerisch-administrative Maßnahmen zielen auf die Planung, den Aufbau und die Pflege eines Qualitätssicherungssystems ab.
- **konstruktiv:**
  Konstruktive Maßnahmen sollen das Entstehen von Fehlern und Qualitätsmängeln von vornherein verhindern. Sie verwenden geeignete Maßnahmen, Methoden und Werkzeuge.

Siehe Lerneinheit 2

- **analytisch:**
  Zu den analytischen Maßnahmen zählen alle Maßnahmen zur Erkennung und Beseitigung von Mängeln und Fehlern („Test") sowie zur Bewertung der Qualität.

Siehe Lerneinheit 3

- **psychologisch orientiert:**
  Zu dieser Gruppe zählen Maßnahmen, in denen der Projektmitarbeiter als Mensch im Mittelpunkt steht (z.B. Gestaltung eines angenehmen Arbeitsklimas).

Zur Verbesserung der Prozessqualität werden Modelle bzw. Verfahren eingesetzt, welche es erlauben, den Software-Entwicklungsprozess auf einen bestimmten, klar definierten Qualitätsstandard zu heben. Dieser kann dann z.T. auch in Form einer Zertifizierung zum Aus-

druck kommen. Beispiele für solche Qualitätsstandards sind:

- ISO 9000
- TQM – Total Quality Management
- CMM – Capability Maturity Model
- SPICE – Software Process Improvement and Capability Determination

ISO 9000

Das ISO 9000-Normenwerk (ab 1985) bildet eine international vereinheitlichte Basis für Qualitätsmanagementsysteme. Richtlinien für die Entwicklung, Lieferung und Wartung von Software finden sich in der ISO 9000-3.

TQM

TQM ist ein gesamtheitlicher Qualitätssicherungsansatz, dessen Entwicklung in Japan ca. ab den 1950er-Jahren begann, beeinflusst von amerikanischen Beratern. Im Mittelpunkt steht der Wunsch des Kunden; TQM weist einen geringeren Grad an Formalisierung auf und baut stark auf der Verantwortlichkeit von Managern und Mitarbeitern auf. Eine signifikante Methode ist das „House of Quality", eine hausförmige Matrix zur Bewertung der Kundenanforderungen.

CMM wurde Mitte der 1980er-Jahre am Software Engineering Institute (SEI) an der Carnegie Mellon University in Pittsburgh entwickelt.

CMM wurde – im Gegensatz zu ISO 9000 bzw. TQM – speziell für den Software-Entwicklungsprozess entwickelt. CMM beschreibt fünf Reifegrade (maturity levels) des Software-Entwicklungsprozesses; je höher die erreichte Stufe ist, desto größer sind Produktivität und Qualität bei der Software-Entwicklung und desto niedriger ist das Risiko. Folgende, aufeinander aufbauende Stufen werden unterschieden:

1. initial
2. repeatable
3. defined
4. managed
5. optimizing

1. initialer Prozess – chaotisch, ad-hoc, unvorhersehbar
2. wiederholbarer Prozess – intuitiv, von Individuen abhängig
3. definierter Prozess – verbesserte, noch nicht vorhersehbare Qualität
4. gesteuerter Prozess – Einsatz von Metriken, kontrollierte Produktqualität
5. optimierender Prozess – permanente Verbesserung durch Rückkopplung der Prozesswerte

Mit Hilfe eines Fragebogens können die Kriterien des Software-Entwicklungsprozesses abgefragt und die Erreichung einer bestimmten Stufe überprüft werden. Ein Wechsel in die nächsthöhere CMM-Stufe kann, je nach geleistetem Aufwand, im Durchschnitt alle zwei Jahre erfolgen, wobei ein Großteil der Software erzeugenden Unternehmen sich in den Stufen 1 bis 3 befinden.

CMMI

Seit 2002 gilt das verbesserte CMMI (I steht für Integrated), welches sich stärker an der ISO 15504 orientiert (siehe folgender Absatz).

SO: International Standards Organisation
ISO 15504

SPICE wird seit 1993 unter dem Dach der ISO entwickelt und soll die bestehenden Ansätze der ISO 9000 und des CMM zusammenführen und vereinheitlichen. Ähnlich wie bei CMM gibt es auch hier Reifestufen (0 bis 5), die Selbstbewertung (Self-Assessment) spielt eine wichtige Rolle.

Üben

1. Erklären Sie anhand eines selbstgewählten Beispiels, warum Qualität und Effizienz konkurrierende Ziele sind.

2. Ordnen Sie folgenden Maßnahmen die richtige Maßnahmenart zu:

	Maßnahme
1	Die gemeinsame Kaffeepause erhöht die Motivation der Mitarbeiter/Mitarbeiterinnen.
2	Im Rahmen der Tests wird die Black-Box-Methode angewendet.
3	Einführung eines Qualitätssicherungssytems in der SW-Entwicklungsabteilung
4	Alle Entwürfe werden mittels Strukturiertem Design (SD) erstellt und dokumentiert.

	Maßnahmenart
a)	planerisch-administrativ
b)	konstruktiv
c)	analytisch
d)	psychologisch orientiert

Skriptenshop

Fallbeispiel Skriptenshop

Erstellen Sie eine Tabelle nach dem folgenden Muster und finden Sie mögliche und im Umfeld des „Skriptenshop"-Projekts realistische Maßnahmen zur Qualitätssicherung. Achten Sie darauf, die Maßnahmen so zu formulieren, dass sie sofort umgesetzt werden könnten.

Schwerpunkt der getroffenen Maßnahme:	Vorschläge für konkrete Maßnahmen:
planerisch-administrativ	• • • ...
konstruktiv	• • • ...
analytisch	• • • ...
psychologisch orientiert	• • • ...

Sichern

Qualität

Qualität ist die Gesamtheit von Eigenschaften und Merkmalen eines Produkts oder einer Tätigkeit, die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse bezieht.

Merkmal

Ein Merkmal ist jene Eigenschaft, die eine quantitative oder qualitative Unterscheidung eines Produktes oder einer Tätigkeit aus einer Gesamtheit ermöglicht.

Qualität von Software (vgl. Kapitel 8, Seite 263)

Die Qualität einer Software wird durch Merkmale wie Benutzerfreundlichkeit, Wartungsfreundlichkeit, Zuverlässigkeit, Funktionserfüllung, Zeit- bzw. Verbrauchsverhalten und Übertragbarkeit beschrieben.

Qualitätskreislauf

In einem Qualitätskreislauf sind, ausgehend von den Qualitätszielen des Software-Produktes, die erforderlichen Qualitätsziele des Entwicklungsprozesses zu definieren und umzusetzen. Anschließend ist die erreichte Qualität des Software-Produkts gegen die ursprünglich gesetzten Ziele zu evaluieren (bewerten). Abweichungen sind bei der Gestaltung des folgenden Qualitätskreislaufs zu berücksichtigen.

Maßnahmen zur Qualitätssicherung

In Software-Entwicklungsprojekten können Maßnahmen zur Qualitätssicherung in den folgenden Formen gesetzt werden:

- planerisch-administrativ
- konstruktiv
- analytisch
- psychologisch orientiert

Qualitätssicherungsmodelle

Qualitätssicherungsmodelle beschreiben den (Software-)Herstellungsprozess sowie Maßnahmen für dessen Evaluierung (Bewertung) und kontinuierliche Verbesserung. Zur Erreichung internationaler Akzeptanz werden sie in Form von Normen beschrieben. Beispiele sind ISO 9000, CMM (bzw. das neuere CMMI), TQM und SPiCE (ISO 15504).

Lerneinheit 2

Konstruktive Maßnahmen zur Qualitätssicherung

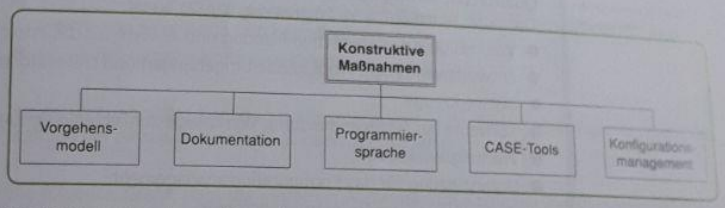
Konstruktive Maßnahmen der Qualitätssicherung dienen zur Schaffung eines qualitätsfördernden Umfelds in einem Software-Entwicklungsprojekt. Dies beinhaltet die Wahl des geeigneten Vorgehensmodells und der passenden Programmiersprache sowie der dazupassenden Tools. In gleicher Weise sind Dokumentation und Konfigurationsmanagement zu planen und durch geeignete Systeme zu unterstützen.

Lernen

1 Übersicht: Konstruktive Maßnahmen

Das Vorhandensein und die Qualität der konstruktiven Maßnahmen bestimmt direkt die Qualität des Software-Entwicklungsprozesses.

Konstruktive Maßnahmen zur Qualitätssicherung



Qualität des Vorgehensmodells

Qualitätsmerkmale von Vorgehensmodellen

- Vollständigkeit
- Modularität
- Systematik
- Allgemeingültigkeit
- Anpassbarkeit
- maschinelle Unterstützung

Wesentlich ist ein Kompromiss zwischen zu geringer Unterstützung und zu starrer Reglementierung. Dieser Kompromiss ist in Abhängigkeit der jeweiligen Prozessumgebung zu finden (siehe auch Kapitel 7, Lerneinheiten 4 und 5).

Qualität der Dokumentation

Qualitätsmerkmale der Dokumentation

- Änderbarkeit
- Aktualität
- Eindeutigkeit
- Identifizierbarkeit
- Normkonformität
- Verständlichkeit
- Vollständigkeit
- Widerspruchsfreiheit

Die oft mangelhafte Qualität von Dokumenten in Software-Entwicklungsprojekten kann verschiedene Ursachen haben:

- Zeitdruck (die lauffähige Applikation steht im Vordergrund)
- Angst der Mitarbeiter vor Offenlegung
- fehlendes „Erfolgserlebnis" beim Dokumentieren
- Anpassen der Dokumentation an die Software-Änderungen ist sehr aufwändig
- Dokumentation wird in Ausbildung zu wenig geschult

Weitere Richtlinien finden Sie im Kapitel 12 „Dokumentation und Abnahme".

Qualitätsbeitrag durch geeignete Programmiersprachen

Qualitätsmerkmale der Programmiersprache

Der Aufwand der Programmierung beträgt nur 20 % bis 30 % des Projektaufwandes. Die Bedeutung der Programmiersprache tritt vor allem in der Wartung zutage.

Qualitätssichernde Konzepte von Programmiersprachen sind:

- Modulkonzept
- Datenkapselung, abstrakte Datentypen
- strukturierter Kontrollfluss
- Datentypenkonzept und Laufzeitprüfungen
- beschreibende Namen
- objektorientierte Programmierung
- Unterstützung von Prototyping

Qualität durch CASE-Werkzeuge

Qualitätssicherung durch CASE-Werkzeuge

Computer Aided Software Engineering (CASE) bietet:

- Werkzeuge zur strategischen Planung und Analyse für das Informationssystem
- Entwurfswerkzeuge, Programmierumgebungen und Generatoren
- Testwerkzeuge
- Konzepte zur Fehlervermeidung, Werkzeuge zur Fehlererkennung und -auswertung
- Wartungswerkzeuge
- Unterstützung für das Konfigurationsmanagement
- Unterstützung des Projektmanagements

CASE-Werkzeuge sollten flexibel hinsichtlich des Vorgehensmodells sein und den gesamten SDLC (Software Development Life Cycle) einschließlich der Wartungsphase unterstützen.

durch Konfigurationsmanagement

Qualitätssicherung durch Konfigurationsmanagement

Das primäre Ziel des SW-Konfigurationsmanagements ist die effiziente Verwaltung der SW-Konfiguration im Life Cycle des Produkts. Dazu wird eine eigene Stelle im Projekt oder Unternehmen eingerichtet, z.B. das „Change Control Board", kurz CCB. Alle Änderungen im Programm, und sei es nur auf Modulebene, müssen vom CCB genehmigt (und anschließend dokumentiert) werden.

Qualitätsmerkmale des Konfigurationsmanagements sind:

- transparente Versionsverwaltung
- Optimierung von Änderungsarbeiten
- kontrollierte Veröffentlichung von Software-Releases
- Vermeidung unkontrollierter Seiten-Effekte durch Änderungen
- jederzeitige Verfügbarkeit bestimmter (auch historischer) Konfigurationen

Im folgenden Abschnitt wird das Konfigurationsmanagement eingehender behandelt.

2 Konfigurationsmanagement

Beim Erstflug der Ariane 5 Trägerrakete im Jahr 1996 verursachte ein Programmteil aus der Ariane 4 aufgrund nicht berücksichtigter Inkompatibilität einen Programmabsturz, Rechnerabsturz und die Explosion der Rakete 40 Sekunden nach dem Start. Unmittelbarer Schaden: circa 850 Mio. Euro.

Software-Konfiguration bezeichnet die Gesamtheit der Software-Elemente, die zu einem bestimmten Zeitpunkt im Life Cycle in ihrer Wirkungsweise und in ihren Schnittstellen aufeinander abgestimmt sind. Ein Software-Element ist entweder der kleinste, für eine Konfiguration unteilbare Bestandteil des Produkts, der eindeutig identifizierbar ist, oder wiederum eine Software-Konfiguration. Eine bestimmte Konfiguration wird durch die Begriffe Baseline, Variante und Revision beschrieben.

Baseline, Variante und Revision

- **Baseline (Bezugskonfiguration):** eine zu einem bestimmten Zeitpunkt im Prozess ausgewählte und freigegebene Konfiguration.
- **Variante:** zwei Software-Elemente, denen eine wesentliche Eigenschaft gemein ist.
- **Revision:** Wenn x und y zwei SW-Elemente sind und y durch Ändern einer Kopie von x erzeugt wurde (y ist Verbesserung von x), so spricht man von einer Revision.

Aufgaben des Konfigurationsmanagements

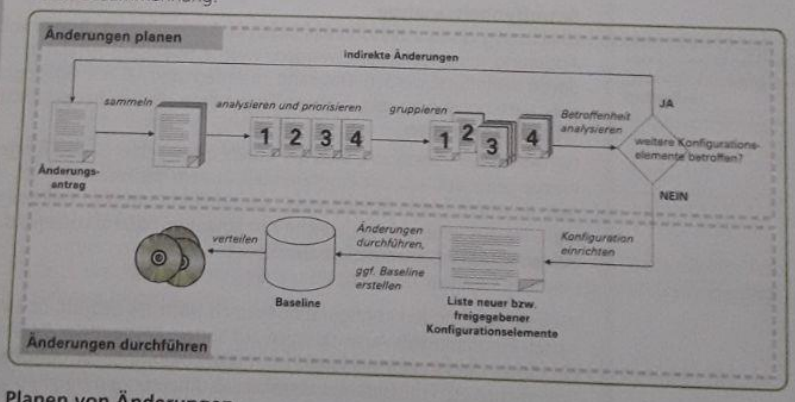
Eine wesentliche Aufgabe des Konfigurationsmanagements ist es, dafür zu sorgen, dass Änderungen nicht willkürlich und an verschiedenen Stellen des Projekts durchgeführt werden, sondern an einer zentralen Stelle gesammelt, geprüft, priorisiert und anschließend freigegeben werden. Organisatorische Voraussetzungen dafür sind:

- schriftlich formulierte Änderungsanträge
- eine Liste aller Konfigurationselemente (Programme, Oberflächen, Dokumentation ...), welche von einer Änderung betroffen sind
- ein aktuell gehaltener Release-Kalender

Für ein wirkungsvolles Konfigurationsmanagement ist der Einsatz eines Repositorys zur Verwaltung der einzelnen Artefakte unbedingt erforderlich.

Die folgende Grafik zeigt die einzelnen Aktivitäten im Konfigurationsmanagement sowie deren Zusammenhang.

Aktivitäten im Konfigurationsmanagement



Änderungsanträge
- sammeln
- analysieren und priorisieren
- gruppieren

Planen von Änderungen

Die Durchführung einer Änderungen beginnt mit den folgenden Schritten:

- **Analyse der Änderungsanträge:** Das Konfigurationsmanagement prüft, welche Priorität die Änderung besitzt, was die Ursache für die Änderung ist, welche Art von Änderung durchzuführen ist und wie aufwändig die Durchführung sein wird.

- **Priorisierung der Änderungsanträge:** Je nach Dringlichkeit der Änderungen ist eine erste Reihenfolge herzustellen. Dabei können z.B. vier Prioritätsstufen zugeordnet werden:

Priorität	Beschreibung	Häufigkeit[1)]
1	Höchste Priorität. Muss unverzüglich durchgeführt werden, hat Priorität selbst über alle sonstigen Aufgaben. Die Arbeit daran erfolgt ohne Unterbrechung bis zur erfolgreichen Durchführung.	0 % bis 2 %
2	Zweithöchste Priorität. Muss vor Aufgaben mit Priorität 3 bzw. 4 bearbeitet werden, unterbricht diese gegebenenfalls.	10 % bis 20 %
3	Priorität der meisten Änderungsanträge. Wird innerhalb normaler Vorbereitungs- und Arbeitszeiten durchgeführt.	50 % bis 70 %
4	Niedrigste Priorität. Sollten erst bearbeitet werden, wenn Sie zu einem Änderungsantrag höherer Priorität „passen". Hochstufen auf Priorität 3 nach gewisser Zeit verhindert „Liegenbleiben" des Antrags.	20 % bis 30 %

[1)] bezogen auf alle Änderungen

- **Festlegen der Wirksamkeit der Änderungsanträge:** Die Festlegung bestimmt, ab welchem Release die Änderungen enthalten sind. Erfolgt in Zusammenhang mit der Release-Planung. Dadurch können durchgeführte Änderungen einem bestimmten Release zugeordnet werden. Abhängig von verschiedenen Faktoren, wie z.B. Priorität, notwendiger Zeitaufwand, Verfügbarkeit notwendiger Ressourcen.
- **Gruppieren der Änderungsanträge:** In diesem Schritt wird geprüft, ob ein Zusammenführen „ähnlicher" Änderungsanträge zur effizienteren Bearbeitung möglich ist. Dies ist der Fall, wenn Änderungen das gleiche Objekt betreffen, gleiche Wirksamkeit und Priorität haben.

Damit die **Behandlungspriorität** möglichst objektiv und nachvollziehbar festgelegt werden kann, gibt es für die Prüfung von Änderungsanträgen Entscheidungsbäume. Anhand des Regelnetzwerks wird z.B. überprüft, wie hoch der Schaden ist, wie kritisch sich der Vorfall beim Nutzer auswirkt, ob es eine kurzfristige „provisorische" Lösung gibt und wie hoch der Lösungsaufwand wäre. Das Ergebnis ist eine Klassifizierung in „Notfall" (Priorität 1) und „Normalfall" (Prioritäten 2 bis 4).

Betroffenheitsanalyse chführen (Auswirkungen ner geplanten Änderung bestimmen)

Betroffenheitsanalyse

Gleichzeitig ist eine Betroffenheitsanalyse durchzuführen: Bereits vor Durchführung einer Änderung muss festgestellt werden, welche (weiteren) Artefakte des Software-Produkts davon beeinflusst werden. Das kann im einfachsten Fall nur ein Programmteil sein, es kann aber auch notwendig sein, mehrere Programme, die Benutzerschnittstelle sowie Dokumentationen anzupassen.

Gehen wir z.B. davon aus, dass der Kunde einen Fehler bei der Eingabe in eine Bildschirmmaske gemeldet hat. Der Programmteil, in dem der Fehler auftritt, konnte lokalisiert werden. In einem ersten Schritt muss nun bestimmt werden, welches Release beim Kunden läuft. Daraus leitet sich ab, welche Konfiguration (intern) vorliegt, d.h. welche Version des (fehlerhaften) Programmteils betroffen ist.

In dieser bestimmten Konfiguration wird als Nächstes geprüft, ob eine Korrektur des fehlerhaften Programmteils Auswirkungen auf andere Bestandteile dieser Konfiguration hat, d.h. bei diesen Änderungen erforderlich macht. Diese indirekt betroffenen Teile (Programme, Dokumente etc.) müssen nun ebenfalls als Änderungsanträge aufgenommen und verwaltet werden.

Alle direkt oder indirekt betroffenen Bestandteile der Konfiguration erhalten nach durchgeführter Änderung neue Versionsnummern (z.B. „+1" hinter dem Versionspunkt).

neue Konfiguration einrichten

Einrichten der Konfiguration

Wurden alle direkten und indirekten Änderungen erfasst, kann eine neue Konfiguration eingerichtet werden. Dazu werden eine Liste der für die Änderung freigegebenen Konfigurationselemente sowie eine Liste neu hinzugekommener Elemente erstellt. Zu jedem Element werden der Verantwortliche sowie die Zugriffsrechte angegeben. Im Fall von Programmteilen wird ein neuer Build-Plan erstellt. Er dient zum automatischen Compilieren und Linken der zur Konfiguration gehörigen Module.

Baseline erstellen

Baselining

Eine gesamte Konfiguration oder Teile einer Konfiguration können bei Bedarf „eingefroren" werden. Eine solche Konfiguration wird Bezugskonfiguration oder Baseline genannt. Dabei sind alle Artefakte zu identifizieren, welche Teil der Baseline werden sollen. Zusätzlich müssen die eingesetzten Entwicklungswerkzeuge – Compiler, Linker – sowie Make-Utility und Build-Pläne miterfasst und „eingefroren" werden. Selbst Hardwarevoraussetzungen, Version des Betriebssystems (mit Service-Packs), Tools etc. sind relevante Informationen für die Beschreibung einer Baseline. So kann sichergestellt werden, dass eine bestimmte Baseline auch zu einem späteren Zeitpunkt wiederhergestellt werden kann.

Zeitpunkte für die Erstellung einer Baseline sind üblicherweise vor der Auslieferung als Release oder bei der Erreichung von Meilensteinen innerhalb des Entwicklungszyklus.

ue Revision verteilen

Verteilung

Die Verteilung kommerzieller Software-Pakete an Kunden – unter Berücksichtigung der Zielkonfiguration beim Kunden – ist ebenfalls Aufgabe des Konfigurationsmanagements. In diesen Bereich fällt sowohl die Wahl des Distributionsmediums (CD oder Download) als auch die Verwaltung der Lizenzierung.

Üben

Die Begriffe in der folgenden Tabelle dienen der Festlegung und Beschreibung bestimmter Konfigurationen. Erklären Sie diese Begriffe.

Baseline	
Variante	
Revision	

Skriptenshop

Fallbeispiel Skriptenshop

1. Erarbeiten Sie eine Entscheidungstabelle oder einen Entscheidungsbaum, die/der festlegt, welche Probleme Ihrer Skriptenshop-Software zu welcher Priorität des Änderungsantrags führen würden.

2. Erarbeiten Sie ein System, nach welchem die Versionen und Revisionen Ihrer Skriptenshop-Programme zu nummerieren sind.

3. Suchen Sie im Internet nach Tools, welche das Konfigurationsmanagement (Revisionsmanagement) in Ihrem Projekt unterstützen würden. Erstellen Sie eine Übersicht der gefundenen Programme (Bezeichnung, Hersteller, Preis, wichtigste Features). Welches Tool würden Sie für Ihr Projekt auswählen?

Sichern

konstruktive Maßnahmen zur Qualitätssicherung

Konstruktive Maßnahmen zur Qualitätssicherung in Software-Entwicklungsprojekten betreffen

- das Vorgehensmodell,
- die Dokumentation,
- die Programmiersprache,
- den Einsatz von (CASE-)Tools,
- das Konfigurationsmanagement.

Wissen

1. Welche konstruktiven Maßnahmen zur Qualitätssicherung im Software-Engineering gibt es?

2. Erklären Sie die Begriffe:
   - Baseline
   - Variante
   - Revision

3. Welche Schritte werden im Konfigurationsmanagement vom Änderungsantrag bis zur Verteilung durchlaufen?

4. Welche Prioritäten für die Bearbeitung von Änderungsanträgen können zugeordnet werden, welche Maßnahmen sind in den einzelnen Prioritätsstufen zu setzen?

5. Welches Ziel hat die Betroffenheitsanalyse im Rahmen des Konfigurationsmanagements?

6. Was versteht man unter Baselining?