The Rational Unified Process

Stefan Fuchsberger
Bahnstr. 4a
65843 Sulzbach
(+49) 6196 759865
Stefan.Fuchsberger@t-online.de

Andreas Remsperger
Königsberger Str. 7
65760 Eschborn
(+49) 6173 64963
AndreasR@appsolut.com

ABSTRACT

Dieses Dokument bietet eine Übersicht über den Rational Unified Process. Der Rational Unified Process ist ein Software Engineering Process, abgebildet durch eine web-basierte, durchsuchbare Knowlegdebase. Der Prozess steigert die Teamproduktivität und bietet „Best Practices“, Entscheidungshilfen und Vorlagen für alle kritischen Stellen eines Entwicklungszyklus. Die Knowledgebase erlaubt es Entwicklern den vollen Nutzen aus dem Industriestandard UML (Unified Modelling Language) zu ziehen.

EINLEITUNG - PROBLEME DER SOFTWAREENTWICKLUNG

Durch die Zunahme immer komplexerer Systeme in unserem Alltag, gewinnt die Entwicklung von Software immer mehr an Bedeutung. Eines der größten Probleme der heutigen Softwareentwicklung ist es, einen Weg zu finden, auf dem sich vorhersagbar qualitativ hochwertige Software herstellen lässt.

Dabei lassen sich folgende markante Punkte ausmachen, die für das Scheitern von Softwareprojekten verantwortlich sind:

- mangelndes Verständnis für die Bedürfnisse und Denkweisen des Kunden,
- inflexible Systeme, die bei Änderung der Randbedingungen in sich zusammenbrechen,
- aufgrund mangelnder Kommunikation während der Entwicklung passen die einzelnen Module eines Systems am Schluss nicht zusammen,
- Projektfehler werden zu spät erkannt und können nur mit großen Kosten behoben werden,
- Schlechte Code-Qualität,
- Mangelnde Performance des Codes resultiert meistens aus der Auffassung, dass es die Hardware schon richten wird,
- Mangelnde Versionsverwaltung führt dazu, dass sich Änderungen nicht zurückverfolgen lassen, warum und von wem sie eingeführt wurden ...

Um nun aber einen vertrauenswürdigen und erfolgreichen Entwicklungsprozess zu gewährleisten, ist es von Nöten, die Ursachen genau zu analysieren und auszuwerten. Hierbei lassen sich folgende acht Punkte ausmachen:

- ad hoc Anforderungsanalysen
- mehrdeutige Kommunikation
- zerbrechliche Architekturen
- überwältigende Komplexität
- Inkonsistentz in Design und Implementierung
- Falsche oder fehlende Risikoeinschätzung
- Unkontrollierbare Auswirkungen bei Änderungen
- Unzureichende Automatisierung

Rational's Best Practice

Aus dem Grunde, qualitativ hochwertigen Software zu produzieren, und den oben genannten Fehlpunkten entstanden die sog. Best Practice, eine Sammlung von Ratschlägen, die einem das Leben und Planen leichter machen sollen. Im einzelnen lassen sich folgende Regelungen dabei ausmachen:

SOFTWARE ITERATIV ENTWICKELN

Die klassische Herangehensweise bei der Softwareentwicklung ist das Wasserfallmodell, bei dem die einzelnen Phasen des Entwicklungsprozesses sequentiell durchlaufen werden.

Eine andere Sichtweise auf den Entwicklungsprozess ist der iterative Entwurf bzw. das Spiralmodell von Barry Boehm. Hier wird die Identifizierung von Fehlern möglichst früh erzwungen. Somit erreicht man eine Korrektur mit vertretbaren Kosten- und Zeitaufwand.

Anforderungen verwalten

Das größte Problem für Systeme sind die sich ständig ändernden Anforderungen. Es muss geradezu erwartet werden, dass sich noch innerhalb des Entwicklungsprozesses die Anforderungen an das fertige Produkt ändern werden. Ebenso ist es nicht möglich alle Anforderungen an ein System schon in der Planungsphase zu erkennen.

Eine sinnvolle Anforderungsverwaltung wird daher folgende Punkte erfüllen:

- Sammlung und Organisation (Dokumentation) der Randbedingungen des Systems
- Veränderungen feststellen und deren Auswirkungen festhalten

- Verfolgen und dokumentieren von Entscheidungen

Komponentenbasierte Systeme verwenden

Ein Softwaresystem macht es erforderlich verschiedene Sichtweisen auf das System zu berücksichtigen. Dabei bildet die Architektur die zentrale Anlaufstele für diese verschiedenen Sichtweisen.

Komponentenbasierte Entwicklung ist eine bewährte Herangehensweise an eine Architektur, da man so die Möglichkeit hat, aus Abertausenden von kommerziellen und nicht kommerziellen Quellen zu wählen (z.B. COM, CORBA, EJB ....)

Software visuell modellieren

Die Modellierung von Zusammenhängen in einem System hilft es besser zu verstehen. Ebenso lassen sich neue Module leichter spezifizieren und dokumentieren.

Die Verwendung einer standardisierten Modellierungs-sprache, wie z.B. UML, hilft, Entscheidungen und Spezifikationen in eindeutiger Art und Weise leicht zu kommunizieren.

Qualität der Software verifizieren

Bei der heutigen Komplexität von Softwaresystemen kann man sich nicht mehr erlauben, Fehler erst nach der Auslieferung / nach Projektabschluss zu finden und beheben zu müssen. Aus diesem Grunde ist es notwendig fortlaufend die Qualität eines Systems sicherzustellen. Unter Qualität eines Softwaresystems versteht man dabei nicht nur die Korrekte Ausführung des Codes, sondern auch die Stabilität bei schlechten Randbedingungen. Ebenso wird die Performance des Systems als ein Maß der Güte herangezogen.

Veränderungen an der Software kontrollieren

Um die Teilprodukte verschiedener Entwickler zu Koordinieren ist es notwendig alle Änderungen am System zu verfolgen und zu dokumentieren. Dies erreicht man am einfachsten durch die Einführung fester Workflows, die es einem ermöglichen Arbeitsabläufe und deren Resultat leichter vorherzusagen.

DER RATIONAL UNIFIED PROCESS - AUFGABEN EINES ENTWICKLUNGSPROZESSES

Ein Softwareentwicklungsprozess hat nach Grady Brooch (nach Kruchten, 1999) vier verschiedene Aufgaben

- Er muss eine Übersicht über die einzelnen Team-aktivitäten liefern,
- Er muss den Zeitpunkt der Fertigstellung bestimmter Teilprodukte angeben,
- Er muss die Aufgaben Einzelner und des ganzen Teams lenken,
- Er muss Kriterien für die Überwachung und Einschätzung der Arbeitsabläufe liefern

DER RATIONAL UNIFIED PROCESS - CHARAKTERISTIKA

Der Rational Unified Process, als ein Software Engineering Process, beinhaltet neben einer ausführlichen Online-Knowledgebase verschiedene Handbücher rund um das Thema Softwareentwicklung und Workflow-Management.

Die Knowledgebase vereinigt einen Tool-Ratgeber, der einen bei der Auswahl der Werkzeuge unterstützen soll, sowie Vorlagen für verschiedene Teilprodukte in einem Softwareentwicklungsprozess.

Als wesentliche Stärken des Rational Unified Process haben sich im Laufe der Zeit herausgestellt, dass er selbst regelmäßig durch Updates ergänzt und den aktuellen Bedürfnissen angepasst wird. Aufgrund seiner Online-Verfügbarkeit in HTML-Form, ist er schnell und von allen Plattformen aus zu erreichen. Eine weiterer wichtiger Pluspunkt für den RUP ist, dass er, obwohl in viele Standardtools (hauptsächlich von Rational) integriert ist, immer noch an die eigentlichen Bedürfnisse des Kunden angepasst werden kann.

Rational stellt als besondere Pluspunkte, neben dem individuell anpassbaren Framework und der strikten Verfolgung der oben genannten Best Practices, für Unternehmen heraus:

- Verteilung aktueller Release - Versionen mittels Intra- / Internet
- Direkte Verfügbarkeit von Schlüsselinformationen mittels integriertet Index- und Metasuchmaschinen
- Leichte Navigation zwischen einzelnen Teilpro-zessen, sowie deren Versionsverwaltung
- Einfache Übernahme von projektspezifischen Verbesserungen in den gesamten Prozess

DER RATIONAL UNIFIED PROCESS - PROZESSSTRUKTUR

In der Prozessstruktur des Rational Unified Process kann man zwei Dimensionen ausmachen. Die erste Dimension ist die sog. "Dynamische Dimension". Sie repräsentiert den Lebenszyklus in dem der Prozess verläuft und wird durch Begriffe wie Zyklus, Phase, Iteration und Meilenstein beschrieben.

Die zweite Dimension in der Prozessstruktur ist die sog. "Statische Dimension". Sie wird durch die logische Gruppierung zusammengehöriger Kernprozesse charak-terisiert. Hier fallen Begriffe wie Komponenten, Aktivitäten, Artefakte und Worker.

Im folgenden möchten wir nun die wesentlichen Merkmale dieser beiden Dimensionen aufzeigen und daran versuchen die Prozessstrukturen zu verdeutlichen.

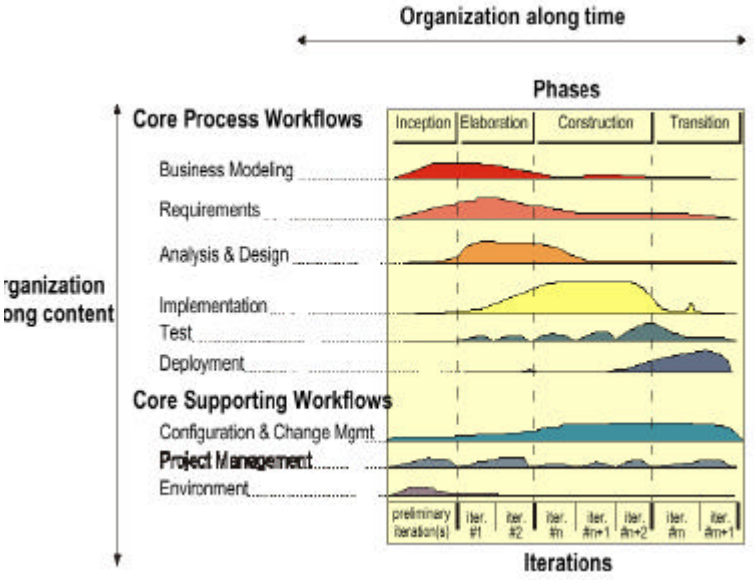

Abbildung 1: Die zwei Prozessstrukturen

Die dynamische Prozessstruktur

Im Gegensatz zum Eingangs angesprochenen Wasserfallmodell ist innerhalb des Rational Unified Process das Iterative Modell implementiert. Dabei entspricht jeder Iteration einem kleinen Wasserfall. Durch dieses Prinzip lassen sich Fehler schnell finden und kostengünstig beheben. Innerhalb einer solchen Iteration findet man folgende Aktivitäten:

- Konzeptionsphase
- Design / Entwurf
- Implementierung / Test
- Freigabe des Produktes

Zusätzlich wird jede Phase einer solchen Iteration mit einem sog. Meilenstein abgeschlossen. An einem solchen Meilenstein lassen sich die gesteckten Ziele und deren Umsetzung messen.

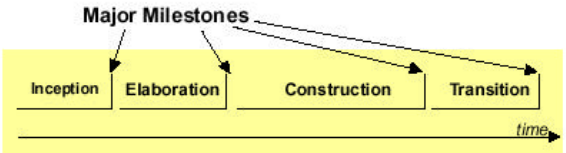

Abbildung 2: Die vier iterativen Phasen

Im Einzelnen möchten wir nun die einzelnen Abschnitte einer solchen Iteration kurz betrachten:

Konzeptionsphase

Hier wird die eigentliche Idee geboren und in ein Produkt überführt. Sprich die Spezifizierung des Endproduktes und seiner Geschäftsvorfälle. Ebenso wird hier der Umfang des Projektes festgehalten. Diese Phase wird durch einen Meilenstein abgeschlossen, der das Ziel des Lebenszyklus wiederspiegelt (Life Cycle Objective – Milestone)

Design / Entwurf

Die Planung der notwendigen Aktivitäten und Ressourcen. Ebenso werden hier die Eigenschaften des neuen Produktes genauer spezifiziert welches in einer Architekturanalyse und schließlich deren Design endet. Diese Phase wird mit einem Architektur - Meilenstein abgeschlossen (Life Cycle Architecture - Milestone)

Implementierungsphase / Test

In der Implementierungsphase wird die Vision in ein handfestes Produkt überführt, sprich implementiert. Die Implementierungsphase wird durch einen "Lauffähige Version" Meilenstein abgeschlossen (Initial Operational Capabilty - Milestone)

Freigabe des Produktes

In der letzten Phase einer Iteration wird das Produkt an die Endbenutzer ausgeliefert. Unter dem Ausliefern eines Produktes versteht man das Training der Benutzer, die Wartung des Produktes durch den Support, sowie Hilfestellungen bei der Migration bestehender Datenbestände. Diese Phase wird durch den Release - Meilenstein abgeschlossen (Release - Milestone), der gleichzeitig den gesamten Zyklus abschließt.

Im Laufe eines Entwicklungsprozesses verschieben sich die Aktivitäten innerhalb der Iterationen. So liegt am Anfang der Schwerpunkt auf Analyse und Design, während sich im späteren Verlauf der Schwerpunkt in Richtung Verkauf und Support verschiebt.

Die statische Prozessstruktur

Wie jeder andere Prozessstruktur beschreibt auch der Rational Unified Process wer wann wie etwas zu erledigen hat. In diesem Zusammenhang werden wir folgende Begrifflichkeit aus dem Rational Unified Process übernehmen.

Unter dem Begriff ***Worker*** versteht man eine Rolle. Diese Rolle schreibt einem Einzelnen oder einer Gruppe von Personen vor wie sie diese Rolle ausführen soll. Dabei kann man ohne weiteres mehreren Rollen in einem Prozess zugeordnet werden.

Der Begriff der ***Aktivität*** beschreibt einen Teil einer Aufgabe, die ein Worker zu erledigen hat. Aktivitäten dienen in der Regel dazu, Informationen neu zu erzeugen (z.B. eine neue Klasse erstellen) oder schon bestehende Informationen zu manipulieren (z.B. einen Absatz in diesem Dokument korrigieren).

Als ***Artefakte*** werden im Rational Unified Process Teile einer Information bzw. die Information selbst bezeichnet.

Artefakt sind der Input für einen Worker und auch wieder der Output jenes selbigen. Der Begriff Artefakt ist eine Besonderheit bei Rational. In anderen Prozessen wird das gleiche meistens als ***work unit*** beschrieben. Im Unterschied zu Rational müssen hier aber nicht die Summe aller Artefakte als fertiges Produkt ausgeliefert werden.

Als ***Workflow*** wird im Ratioal Unified Process eine Abfolge von Aktivitäten bezeichnet. Workflows koordinieren Worker und deren Aktivitäten anhand der Produkte.

Die nachfolgende Grafik verdeutlicht den Zusammenhang zwischen diesen vier Begriffen.

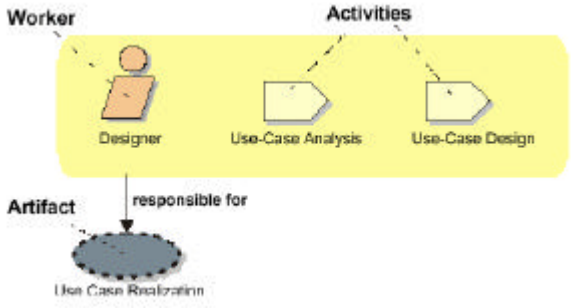

Abbildung 3: Worker, Aktivitäten und Artefakte

DER RATIONAL UNIFIED PROCESS - EIN ARCHITEKTURORIENTIERTER PROZESS

Architektur ist das, was übrigbleibt, wenn man nichts mehr wegnehmen kann und trotzdem das System noch versteht und auch noch versteht, wie es arbeitet.

Bei der Wahl der Architektur eines Softwaresystems legt der Rational Unified Process besonderen auf das Zusammenspiel der einzelnen Komponenten.

Dieses wird dadurch deutlich, das kleine Struktur-komponenten, also z.B. eine Klasse, genauso wie ein fertiges Teilprojekt, z.B. ein Debugger, behandelt werden. Der Architekturbegriff von Rational spiegelt ganz deutlich den Nutzen von komponenten- oder modulbasierten Systemen wieder.

DER RATIONAL UNIFIED PROCESS - EIN USE-CASE GESTEUERTER PROZESS

Der Rational Unified Process beinhaltet neben seinem Schwerpunkt auf der Architektur einen zweiten großen Schwerpunkt, nämlich die Use-Cases. Diese simplen Modelle sollen dazu dienen, einen roten Faden durch das Projekt zu spinnen. Neben dem roten Faden sollen die Use-Cases es ermöglichen, leichter festzustellen, wie das System das tut, was es tut. Neben diesen reinem Verständnisproblemen kommt den Use-Cases noch die Bedeutung zu, dass man mit ihnen, die Modellierung des Objektmodells viel leichter erledigen kann.

Use-Cases stellen eine wichtige Verbindung zwischen den einzelnen Teilen der Softwareentwicklung, z.B. Design und Test, dar. Da der Rational Unified Process neben seiner Architektursicht stark durch Use-Cases geprägt ist, bilden die für das System definierten Use-Cases eine lebenswichtige Grundlage.

Der Rational Unified Process gibt zwei Definitionen für Use-Cases:

1. Ein Use-Case ist eine Sequenz von Aktivitäten, die für einen bestimmten Akteur ein verwertbares Ergebnis liefern.
2. Ein Akteur ist eine Person oder eine Sache, welche mit dem System von Außen in Kontakt tritt.

Hinter jedem Use-Case steckt ein Stück Funktionalität des Gesamtsystems. Folglich wird die Funktionalität des ganzen Systems durch die Summe der einzelnen Use-Cases gebildet.

Ereignisfluss in einem Use-Case

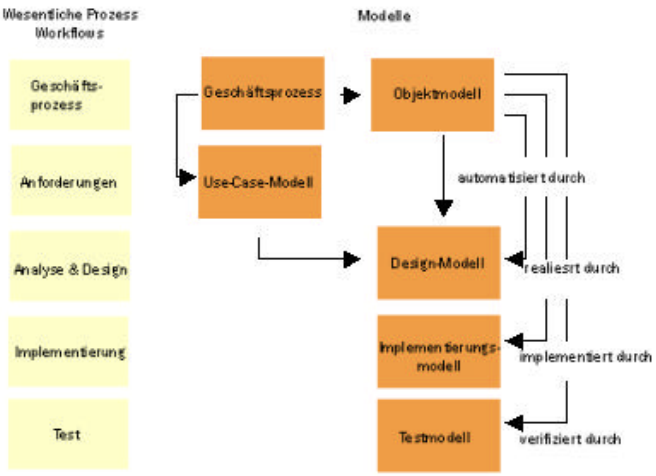

Abbildung 4: Use-Cases

Bei kompletter Betrachtung eines Use-Cases stellt man fest, dass der zu durchlaufende Weg von folgenden drei Faktoren abhängt.

Eingaben des Akteurs

Der Akteur kann Auswählen wie er mit dem System in Kontakt treten möchte

Interner Systemstatus

Der interne Systemstaus ist von entscheidender Bedeutung für das Ausführen bzw. Anbieten von Funktionen.

Time-Outs und Fehler

Wenn z.B. die Anmeldung an einem anderen System scheitert.

Am Anfang eines Projektes, wenn man die Use-Cases definieren möchte, ist es meist sinnvoller mit Szenarien anzufangen, da sich aus diesen dann leicht die Use-Cases ableiten lassen.

Das Use-Case-Modell spiegelt alle Funktionen eines Systems wider. Es dient sozusagen als Vertrag zwischen den Entwicklern und den Kunden. Das Use-Case-Modell umfasst die gesamte Komplexität eines Systems mit all seinen Funktionen und Akteuren.

Nichtfunktionale Beschreibungen komplementirren das Use-Case-Modell. Dazu zählen zum einen Aspekte, die sich nicht durch einen Use-Case beschreiben lassen und zum anderen Aspekte, die für mehrere oder alle Use-Cases gelten.

Im folgenden möchten wir kurz einige Punkte nennen, in denen die Use-Cases immer wieder in Vorschein treten:

- Use-Cases sind das Ergebnis der Anforderungs-analyse. Sie beschreiben hier das System aus Sicht der Endanwender.
- In der Analyse und Designphase lassen sich aus den Use-Cases die einzelnen Klassen zu Implementierung ableiten.
- Durch die Umsetzung der Use-Cases in das Designmodell wird sichergestellt, dass die Dynamik des Systems erhalten bleibt.
- Am Ende dienen die Use-Cases schließlich zur Validierung des Systems. Aus ihnen werden die Testfälle abgeleitet.
- Ebenso lassen sich Use-Cases auch zur Dokumentation eines Systems heranziehen.

DER RATIONAL UNIFIED PROCESS - WORKFLOWS

Im Folgenden möchten wir auf die wichtigsten Workflows im Rational Unified Process eingehen. Dabei orientieren sich die einzelnen Abschnitte an dem Schema Zweck, Worker und Artefakte.

Der Rational Unified Process kennt in seiner Grundform neun verschieden Workflows. Diese werden in sechs Core-Workflows und drei Supporting Workflows unterschieden. Zu den Core-Workflows zählen

- Business Modeling Workflow,
- Requirenment Workflow (Konzeption),
- Analysis & Design Workflow (Design / Entwurf),
- Implementation Workflow (Entwicklung),
- Test Workflow (Test) und
- Deployment Workflow (Freigabe des Produktes),

während man unter den Supporting Workflows

- Projekt Management Workflow,
- Configuration and Change Management Workflows und
- Environment Workflow versteht.In dieser Betrachtung wollen wir uns nur auf die vier letzten Core Workflows konzentrieren, da diese sich in erster Linie mit der eigentlichen Softwareentwicklung auseinander setzten.

Requirenment Workflow

Hier wird eine Vision des Systems beschrieben und in ein Use-Case Modell umgewandelt. Aus den einzelnen Bedürfnissen aller Personen, die ein Interesse an dem System haben, wird das Visionsdokument entwickelt, dass allgemein die Fähigkeiten des System beschreibt.

Der Requirenment Workflow verfolgt folgende Ziele:

- Was soll das System leisten, Übereinstimmung beim Kunden und Entwickler. Entwickler erhalten so ein besseres Verständnis über die Anforderungen
- Funktionalität des Systems definieren
- Grundlage für die Planung der technischen Inhalte
- Grundlage für Kosten und Zeitbedarf
- Benutzerschnittstelle definieren

Um die oben genannten Ziele zu finden, sind in der Regel folgende Personen beteiligt:

- System-Analyst
  Leitet und koordiniert die Anforderungsanalyse, indem er die Systemfunktionalität umreißt und einschränkt
- Use-Case-Specifier
  Legt die Funktionalität des Systems mittels der Beschreibung von Use-Cases fest
- Architekt
  Der Architekt sucht in den Use-Cases nach den architekturrelevanten Anwendungsfällen, um diese dann in seiner Architektur zu berücksichtigen
- User-Interface-Designer
  Leitet und koordiniert den Entwurf und die Entwicklung einer ersten User-Schnittstelle
- Requirenments-Viewer
  Koordiniert die einzelnen Tätigkeiten der Leute. Außerdem ist der für alle Endprodukte des Requirenment Workflows verantwortlich.

Im folgenden die wichtigsten Schlüsselinformationen und Ergebnisse zusammengefasst:

- Stakeholder-Needs
  Eine allgemeine Zusammenfassung der Bedürfnisse der Endanwender
- Vision-Document
  Eine allgemeine Beschreibung der Vision des Produktes
- Use-Case-Modell
  Möglichst ein in UML (oder vergleichbares) modelliertes Modell von Anwendungsfällen
- Supplementare-Specifications
  Anforderungsbeschreibungen die nicht in Use-Cases abgebildet werden können (z.B. Sicherheitsfragen)

- Requirenments-Attributes
  Eine detaillierte Aufstellungen von Anforderungen und deren Abhängigkeiten
- User-Interface-Prototype
  Prototyp der Benutzerschnittstelle

Analysis & Design Workflow

Während der Analyse werden die gesammelten Anforderungen in eine Spezifikation übertragen, die Zeigen soll, wie das System funktioniert und implementiert werden soll. Dabei müssen die einzelnen Anforderungen in eine Form gebracht werden, die ein Softwareentwickler versteht, also in Form von Klassen, Klassenmodellen und Teilsystemen.Während des Designs ist es das Ziel das System mit allen Randbedingungen in Einklang zu bringen. Dabei geht es darum den Entwurf des Systems zu optimieren und die Einhaltung aller Anforderungen sicherzustellen.

Um diese Ziele sicherzustellen spielen folgende Personen eine wichtige Rolle:

- Architekt
  Der Architekt leitet und koordiniert technische Aktivitäten während des Projektes. Er sucht nach grundlegenden Mustern, Schlüsselmechanismen und Modellierungskonventionen. Im Gegensatz zum Rest hat der Architekt eher einen Gesamtüberblick über das System
- Designer
  Er definiert die Verantwortlichkeiten, Operationen, Attribute und Beziehungen zwischen verschiedenen Klassen. Ebenso bestimmt er wie die Klassen auf die Implementierungsumgebung abgestimmt werden.
- Database-Designer
  Falls für das System eine Datenbank benötigt wird, so entwirft der Database Designer diese und setzt sie auf der entsprechenden Zielplattform um.
- Architecture Reviewer und Design Reviewer
  Optionale Spezialisten, die die Schlüsselartefakte (Modell, Architektur) überprüfen

Im folgenden sind noch einmal die Artefakte für den Analysis & Design Workflow zusammengestellt.

- Design-Modell
  Skizze des zu konstruierenden Systems
- Software-Architecture-Document
  Beschreibt die verschiedenen Architektursichten auf das System

Implementierungs – Workflow

Während der Implementierung wird ein System nahezu vollständig implementiert und aus einzelnen Subsystemen zusammengesetzt. Während der Implementierung ist Testen lediglich auf die einzelnen Komponenten durch den Entwickler selbst beschränkt, Ein Integrationstest, also ein Test ob alle Komponenten miteinander arbeiten (was sie ja aufgrund der zuvor durchgeführten Arbeiten tun sollten) wird erst im eigentlichen Test – Workflow durchgeführt.

Als die vier wesentlichen Ziele kann man nennen:

- Quellcodeorganisation
  Der Quellcode eines Systems wird in Form eines Schichtenmodells von Subsystemen vorgenommen
- Implementierung von Klassen und Objekten
  In die Implementierung ist das Erstellen von ausführbaren Laufzeitkomponenten eingeschlossen
- Testen der einzelnen Einheiten
- Integration der einzelnen Komponenten und Subsysteme zu einen ausführbaren System

An der Implementierung eines Systems sind im wesentlichen

- Implementierer
  Implementierung und Test der einzelnen Komponenten
- Systemintegrator
  Erzeugt einen Build (=operationale Version des Systems, beihaltet einen Teil der Funktionen, die im fertigen System enthalten sind)
- Architekt
  Legt die Struktur der Implementierung mit Subsystemen und Schichten fest
- Code-Gutachter
  Prüft den Code auf Qualität und Übereinstimmung mit dem Projektstandard

beteiligt.

Im Nachfolgenden sind wieder die einzelnen Schlüsselartefakte zusammengestellt.

- Implementierungssubsystem
  Hier findet man eine Ansammlung von einzelnen Komponenten und kleineren Teilsystemen. Es wird zur Strukturierung des Implementierungssystems genutzt, indem man es in kleine Teile zerlegt.
- Komponente
  Eine Komponente ist ein Stück Software (Quellcode, Binär) oder eine Datei die Informationen zum Projekt enthält (Readme oder Makefile). Eine Komponente kann auch eine Zusammenfügung mehrerer einzelner Komponenten zu einem kleinen Teilsystem
- Integrationsplan
  Hiermit wird die Reihenfolge der Entwicklung der einzelnen Komponenten festgelegt. Des weiteren spezifiziert es die Build-Reihenfolge bei der Integration eines Systems

Test Workflow

Sind und Zweck des Testen eines Systems ist es, eine Aussage darüber zu bekommen, wie weit man sich seinen Qualitätsanforderungen genähert hat. Der Rational

Unified Process beschränkt sich lediglich auf diese Aussage.

Um eine solche Aussage treffen zu können, benötigt man ein Testmodell. Dieses Modell kann man leicht aus den Use-Case Szenarien ableiten und es wird in der Regel aus folgenden Komponenten bestehen:

- Testfälle
  Als Testfälle bezeichnet man die Menge aller Testdaten, Ausführungsbedingungen und erwartete Testergebnisse für ein bestimmtes Testobjekt. Testfälle können aus den Use-Cases, den Design oder aus dem Quellcode abgeleitet werden.
- Testprozeduren
  Testprozeduren sind detaillierte Instruktionen zur Ausführung und zur Evaluierung der Testergebnisse
- Testskripte
  Im Idealfall sind dies maschinenlesbare Anweisungen, um den Testprozess zu automatisieren.

Für die Testphase sind im wesentlichen zwei Personengruppen zu nennen:

Test-Designer

Der Test-Designer entwirft die eigentlichen Tests. Dazu gehören das Festlegen der Testfälle, die Implementierung der Testscripte, die Evaluierung der Testergebnisse, Testberichte und Testeffektivität.

Systemtester

Der Systemtester ist für den Aufbau des Testszenarios sowie für die eigentliche Durchführung des Testes verantwortlich.

Nachfolgend haben wir die wesentlichen Artefakte und deren Beschreibung zusammengefasst:

- Testplan
  Der Testplan beschreibt die Ziele des Testens. Er enthält die Teststrategie sowie die notwendigen Ressourcen
- Testmodell
  Enthält die Testfälle, Testprozeduren und –scripte
- Auslastungsmodell
  Das Auslastungsmodell dient dazu Aussagen über die Performance zu erlangen
- Fehler
  Fehler die als Ergebnisses eines Tests auftreten werden als Änderungsanforderung eingepflegt

Für jeden Testtyp (Performancetest, Funktionstest,...) kann man den eigentlichen Testworkflow in folgende 5 Phasen zerlegen:

Planung

Der Testdesigner identifiziert die eigentlichen Testanforderungen, Ressourcen und Strategien. All dies wird in einem Testplan dokumentiert, der wiederum auch den Umfang des Tests spezifiziert

Design

Während der Designphase analysiert der Testdesigner das Testobjekt und entwirft anhand der gewonnen Informationen das Testmodell. Soll ein Performancetest durchgeführt werden, so wird ein Lastmodell entworfen.

Implementierung

In der Implementierung programmiert der Testdesigner die zuvor festgehaltenen Prozeduren auf. Werden bestimmte Testreiber benötigt, so werden diese gemeinsam mit dem Designer und den Entwicklern erarbeitet.

Ausführung

Der Tester führt den Test anhand der Testprozedur manuell oder automatisch anhand eines Testskriptes durch. Wenn alle Testläufe beendet sind, fasst der Tester die Tests zusammen, um den Erfolg eines Testlaufes zu bestimmen. Anschließend werden alle gefunden Fehler dokumentiert und zur Behebung weitergereicht.

Bewertung

Am Schluss wird der Testlauf noch statistisch bewertet, um eindeutige Aussagen über Qualität des Produktes, des Tests und dessen Effizienz machen zu können.

ANHANG

Literaturangaben

1. Kruchten, Philippe: Der Rational Unified Process – Eine Einführung; Addison Wesley Longman, 1999; ISBN: 3-8273-1543-3

Abbildungsverzeichnis