Handelshögskolan
Göteborgs universitet
Institutionen för informatik

Systemutvecklingsmetoder vid fem IT-företag i Göteborg

Med detta arbete har vi kartlagt olika utvecklingsmetoder, såväl iterativa som sekventiella för att ge större kunskap inom ämnet. Arbetet behandlar Binomens Luxus ®, Cap Geminis Perform, Ericsson Mobile Data Designs Darwin, Rationals Unified Process som används på Frontec och Ericsson-Hewlett Packard Telecommunications. Uppsatsen redogör för tankarna och strukturerna inom dessa metoder och beskriver även vad iterativt och sekventiellt arbete innebär.
Frågor som besvaras är bl.a. vilka metoder som används, hur pass detaljerade de är, om företagen tagit fram dem själva eller köpt in dem, om man kan se några paralleller mellan företagens storlek, ålder och om de jobbar marknadsnära eller kundnära och metoden de använder sig av.
Dessa frågor har besvarats genom att vi har gjort intervjuer på företagen och därefter fördjupat oss i deras metoder.

Susanna Haikara
Sofia Larsson
Bodil Näsström
Examensarbete I 10 p.
ADB-programmet våren 1999

Handledare: Birgitta Ahlbom

Innehållsförteckning

1 Inledning

1.1 Problembakgrund

Fram till 60-talet var datortekniken i sig så ny att det då inte fanns några systemutvecklingsmodeller. Under 70-talet arbetade man enligt utdataorienterade metoder, dvs. att man inriktade sig på vad systemet skulle producera.
Metoder för att bygga system skapades från början enbart för stora företag där man normalt hade en god framförhållning och en systemutredning därför kunde ta ganska lång tid. I slutet av 70-talet fick dock även mindre företag möjlighet att skaffa egna datasystem varför en hel del nya metodansatser började diskuteras.
Krav på detta slag av systemering är framför allt lågt pris och kort utvecklingstid. Systemeringen avser att skapa en modell av ett avgränsat verksamhetsområde, ett område inom vilket det finns problem som man skall lösa. Modellen skall vara så formaliserad att den kan ligga till grund för en eventuell datorrealisering. Modellen har även som syfte att utgöra ett kommunikationsmedel för de parter som är berörda av systemutredningen.
Datorer har kommit att spela en allt större roll i systemutvecklingsprocessen. De hjälper till att snabbare klara av att producera dokumentation, att snabbare ta fram konsekvenser av förändringsförslag. Diverse CASE-verktyg har utvecklats för att stödja systemutvecklingsarbetet.

1.2 Syfte

Detta arbete syftar till att kartlägga olika utvecklingsmetoder, såväl iterativa som sekventiella för att ge större kunskap inom ämnet.

1.3 Problemdefinition

- Vilka metoder används?
- Hur detaljerade är metoderna?
- Använder sig företagen av en redan existerande metod eller har de utarbetat en egen?
- Vad tar företagen med i beräkningarna när de skall bestämma sig för en viss metod?
- Kan man se några paralleller mellan storleken och ålder på företaget och metoden de använder sig av?
- Kan man se några paralleller om företagen arbetar kundnära[1] eller marknadsnära[2] och deras metod?

[1] Med kundnära avses företag som utvecklar system och program för en specifik kund.
[2] Med marknadsnära menar vi här företag som utvecklar standardsystem för marknaden, dvs ingen specifik kund.

1.4 Metod

För att besvara våra frågor har vi läst litteratur runt ämnet systemutveckling. Det har varit svårt att få tag i relevant litteratur angående specifika metoder. Däremot har vi hittat en hel del bra böcker om systemutveckling i allmänhet. För att få reda på mer om moderna metoder som används i praktiken fick vi istället besöka och göra intervjuer på några av Göteborgs ledande IT-företag.
Frågorna vi ställde till företagen återfinns i bilaga1.

Vi kontaktade ett flertal företag och talade med de som visade intresse.
Vi har besökt Pär Lagström på Frontec, Staffan Ehnebom på Ericsson Hewlett Packard Telecommunications, Anna Börjesson på Ericsson Mobile Data Design, Jan Windahl på Binomen och Jan Lindqvist på Cap Gemini. Vi har även fått ta del av deras metoder och det är dessa vi har fördjupat oss i.

1.5 Disposition

Uppsatsen börjar med en inledande teoretisk del som skall ge bakgrundsfakta till vårt arbete. Här beskrivs de olika tillvägagångssätten vid systemutveckling. En del handlar även om UML (Unified Modelling Language) och dess skapare.
Därefter följer en presentation av företagen vi har valt att titta på samt deras metoder.
De sista sidorna tillägnas resultatet, jämförelser och våra egna slutsatser.

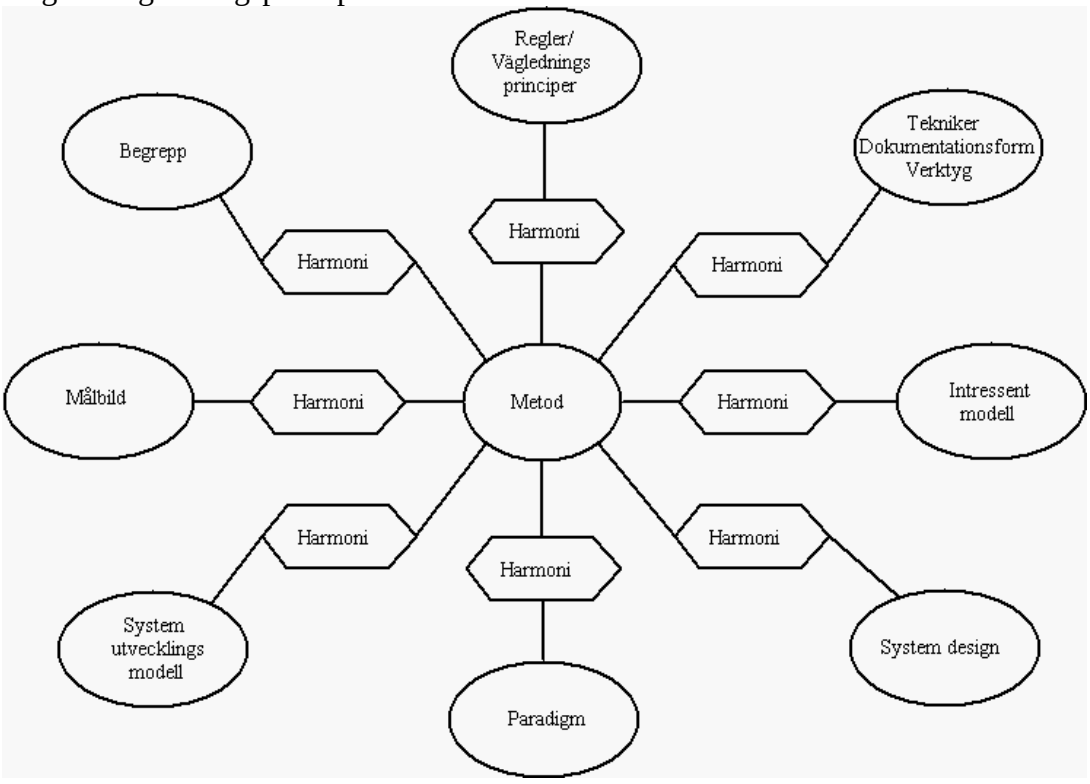
2 Teoretisk bakgrund

2.1 Vad är en systemutvecklingsmetod?

Systemering avser att skapa en modell av ett avgränsat verksamhetsområde och att utgöra ett kommunikationsmedel för de parter som är berörda av systemutredningen.
En metod är det konkreta tillvägagångssättet för att realisera en modell. Metoden är ett medel för att säkerställa kvalitén på systemet. Den ger även en gemensam notation som är viktig för att se till att alla som är involverade i projektet talar samma språk. Användandet av metoden ser också till att alla deltagarna i systemutvecklingsarbetet får en god överblick över systemet och kan följa arbetets gång.

Thanos Magoulas hävdar att om en metod skall vara tillämpningsbar på ett företag måste den harmoniera med:

- begrepp
- målbild
- systemutvecklingsmodell
- paradigm
- systemdesign
- intressentmodell
- tekniker, dokumentationsform, verktyg
- regler/ vägledningsprinciper



Figur 1

Inom systemutveckling brukar metoderna innehålla följande faser och beskrivningar på hur man skall göra.

Analys: I den första fasen görs en verksamhetsanalys, eller en ”prestudy”, som ligger till grund för kravspecifikationen som talar om vad system skall göra och inte göra. Slutprodukten av analysen är kravspecifikationen.

Design: Under designfasen byggs en övergripande arkitektur och de olika delkomponenterna specificeras. Det är under den här fasen som själva byggritningen för systemet arbetas fram t.ex. dataflödesdiagram, relationsscheman, strukturdiagram osv.

Realisering: I realiseringsfasen skapas systemet fysiskt efter de ritningar och specifikationer som togs fram under designfasen.

Implementering: Systemet implementeras hos slutanvändarna.

Test: Många olika typer av test utförs för att säkerställa att systemet gör det som det är skapat för att göra dvs. uppfyller de krav som togs fram under analysfasen.

Underhåll: När systemet tagits i bruk måste det underhållas eftersom nya krav uppstår under användandet, fel hittas, vissa uppdateringar kan behöva göras och det kan finnas behov av att lägga till ny funktionalitet.

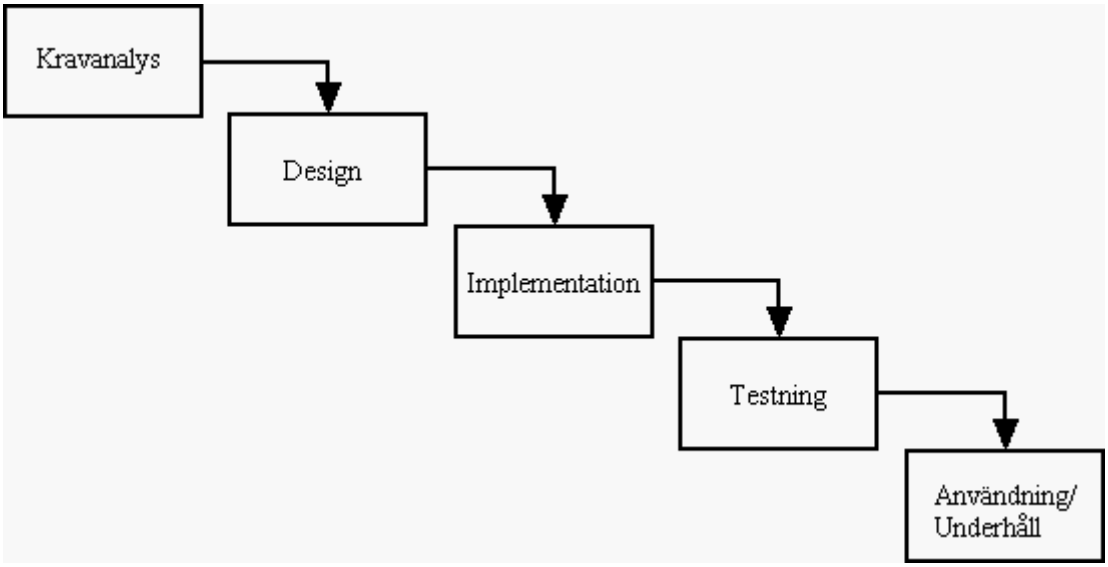
Detta är grundstegen i alla systemutvecklingsmetoder som vi har tittat på. Hur de utförs och i vilken ordning varierar från metod till metod vilket vi kommer att beskriva längre fram.

2.2 Varför behövs metoder?

Olika metoder skiljer sig mycket från varandra och strävar ofta mot olika mål.
Exempel på mål:

- Få fram exakta krav för informationssystemet
- Effektivisera utvecklingsprocessen
- Få fram ett system i tid till rimlig kostnad
- Få fram bra dokumentation och ett system som är lätt att underhålla
- Få fram eventuella ändringar som behöver göras så tidigt som möjligt i utvecklingsprocessen
- Få fram ett system som alla berörda blir nöjda med

2.3 Vattenfallsmodellen



Figur 2

Vattenfallsmodellen är *sekventiell*. Detta innebär att den består av ett antal olika faser och att man innan man övergår till nästa fas skall ha avslutat den föregående.
Den ligger till grund för många av de metoder som existerar idag.
De huvudsakliga faserna i modellen är:

- **Kravanalys och kravspecifikation:** I samarbete med kunden görs en analys av vad systemet skall göra och vilka problem som måste lösas. Detta sammanställs sedan till en kravspecifikation.
- **Design:** En övergripande arkitektur byggs och olika delkomponenter specificeras.
- **Implementation och enhetstestning:** De olika delkomponenterna utvecklas och varje enhet testas.
- **Integration och testning:** De olika delarna sätts samman och testas tillsammans för att verifiera att det fungerar.
- **Användning och underhåll:** Systemet installeras hos kunden och tas i användning.

Nyupptäckta fel rättas och systemet anpassas ytterligare till att passa kundens krav.
Tanken är att man skall passera de olika faserna i följd.

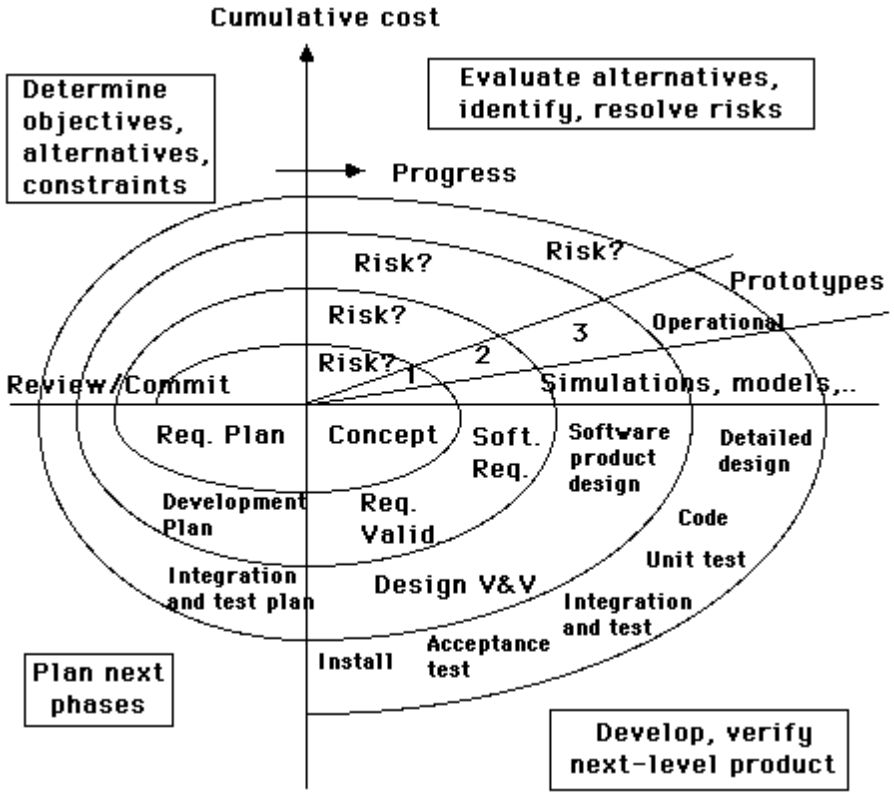
2.4 Iterativ modell

Iterativa modeller går ut på att man successivt bygger fram systemet. Vid iteration kan man arbeta med mer än en fas åt gången medan man under sekventiellt tillvägagångssätt alltid gör färdigt en fas innan man påbörjar nästa. Man kan under vilken fas som helst gå tillbaka ett eller flera steg för att sedan upprepa stegen (till skillnad från sekventiell utveckling där man oftast gör färdigt hela vägen innan man kan gå tillbaka). Under iterativt arbete har man inte samma kontroll, som under sekventiellt arbete, på var man är i utvecklingen dvs. att man sällan har milstolpar som talar om vad som skall vara färdigt när, utan mer riktlinjer för vad som bör ha åstadkommits vid en viss tidpunkt. Därmed är det inte sagt att det måste vara färdigt eftersom man i iterativt arbete aldrig blir helt färdig eftersom man hela tiden förbättrar och bygger på systemet.

[3]En iteration är en komplett utvecklingsloop som resulterar i en release av en körbar produkt, en del av den slutliga produkten som är under utveckling, som alltså växer inkrementellt från iteration till iteration för att till slut bli det färdiga systemet.
Ett iterativt arbete innebär att man kan gå tillbaka och ändra i ett användningsfall utan att det påverkar ett annat. I och med att man skapar användningsfall kan man dessutom skapa ett gränssnitt för varje användningsfall. Detta sker tidigt i utvecklingsprocessen och det är bra att kunna visa upp för kunden vad denne har att vänta sig. Dessutom kan man rätta till små problem och önskemål mycket tidigt i utvecklingsprocessen.
För att grafiskt se hur faserna följer varandra i en iterativ modell se figur 13.

[3] Enligt Rational Software. För mer information gå in på Rationals hemsida http://www.rational.com

2.4.1 Boehm´s spiralmodell



Figur 3

Boehm tog 1988 fram en alternativ processmodell, spiralmodellen, som tar hänsyn till eventuella risker i utvecklingsprocessen.
Varje varv i spiralen motsvarar en fas i utvecklingsprocessen. Dessa faser är inte förutbestämda, utan ledningen för projektet måste besluta hur projektet ska struktureras i olika delar. Varje varv eller fas delas i fyra sektorer:

- **Målsättning:** specifika mål för projektfasen definieras, liksom begränsningar i den övergripande utvecklingsprocessen, och en detaljerad projektplan dras upp. Risker för projektet identifieras och alternativa strategier beroende på dessa risker planeras.
- **Riskbedömning och riskreducering:** för varje identifierad risk görs en detaljerad analys, och åtgärder vidtas för att reducera risken.
- **Utveckling och validering:** en utvecklingsmodell väljs för systemet, t ex vattenfallsmodellen. Valet av modell bör utgå från de identifierade riskerna.
- **Planering:** projektet ses över och beslut fattas om man ska gå vidare med nästa fas (nästa varv i spiralen). Planer för nästa fas dras då upp.

Boehm föreslår ett standardformulär för varje varv eller fas som bör innehålla följande delar:

Målet vilket beskriver vad man vill åstadkomma med analysen
Begränsningar som kan minska möjligheterna till att lyckas med projektet
Alternativ till olika sätt att uppnå målen
Risker som är tänkbara med de alternativ man valt
Riskresolution innebär strategier för att minimera riskerna
Resultat är det som åstadkommits av arbetet under fasen
Planering för hur man går vidare till nästa fas
Åtagande innebär ett ledningsbeslut för hur och om man ska fortsätta

2.5 Inkrementell utveckling

För att utveckla stora programsystem har man under ett antal år nu förordat vad man kallar en inkrementell programframtagning. Med detta menas att man istället för att göra färdigt ett helt programsystem med alla beställarens krav från början så koncentrerar man sig först på de mest centrala delarna och gör ett system som bara klarar dessa.

2.6 Objektorienterad modellering

Objektorientering i sig är inte en metod utan snarare ett synsätt på hur man skall gå till väga för att modellera.
När man arbetar objektorienterat delar man in den, för systemet, väsentliga delen av verksamheten i objekt. Dessa objekt har tillstånd, beteende och attribut.
Attributen är objektets egenskaper, objektet ’person’ har t.ex. attributet ’kön’. Ett av objektets attribut, eller en sammansättning av attribut, måste vara unikt för att man skall kunna identifiera instansen av klassen. Tillståndet är värdena på attributen. Beteende beskriver vilka handlingar objektet kan utföra.
Genom att gömma både data och programkod inom objektet (inkapsling) uppnås både stabilitet och portabilitet. Detta innebär att objekten känner till varandras yttre egenskaper och kan därmed skicka och ta emot information sinsemellan. Däremot är ett objekts inre egenskaper skyddade från insyn.

2.7 UML - Booch, Rumbaugh och Jacobson

UML (Unified Modelling Language) är ett standardiserat visuellt modelleringsspråk, men som ändå på ett smidigt sätt tillåter anpassningar till olika organisationer, utvecklingsprocesser och verktyg. UML beskriver samtliga vanliga begrepp och aspekter av objektorientering, men är samtidigt enkelt att anpassa och man kan själv bestämma hur mycket av UML man vill använda. UML har kommit att bli en standard både för att kommunicera modeller till människor och för att utveckla modeller med datorverktyg.

UML är en sammansmältning av de metoder som skapats av de tre ledande guruerna inom modellering: Grady Booch (Booch-metoden), James Rumbaugh (OMT) och Ivar Jacobson (Objectory), som numer alla tre är anställda på programvaruföretaget Rational.

Samtidigt som Rational presenterar UML som världsvinnande metod släpper de version 4.0 av sin objektorienterade utvecklingsmiljö Rational Rose.

1995 bestämde sig OMG (Object Management Group) för att det var dags att ta fram en standard för objektorienterade metoder. Grady Booch och James Rumbaugh satt vid det här laget båda på Rational—som gör utvecklingsverktyget Rose—och försökte bygga ihop sina standarder, Booch och OMT, till en.
Sommaren 1995 kom Ivar Jacobson med och de blev "the three amigos".

Rational har satsat stort på att få sina skyddslingar att enas. Utvecklingsverktyget Rational Rose, har tidigare haft stöd för modellering med OMT och Booch. I den nya versionen, Rose 4.0, finns även stöd för UML.

Dessa tre "gurus" har alltså förenats för att etablera ett standardiserat modelleringsspråk som ett första steg i att få ett slut på de sk "metodkrigen". De har också presenterat en "Unified Process", dvs en grundläggande process för hur själva utvecklingsarbetet ska gå till (vad ska göras, när ska det göras, hur ska det göras, och varför ska det göras), som heter RUP (Rational Unified Process).
UML togs officiellt i bruk som standard av OMG 1997.

En mycket intressant mekanism som införts i UML är begreppet stereotyper. En stereotyp är en anpassning av ett visst modellelement, där utvecklaren själv kan definiera ytterligare semantik (betydelse) till ett redan existerande modellelement. En stereotyp kan även tilldelas en egen ikon så att den har ett unikt utseende i modellen. Genom stereotyper kan UML anpassas till en viss typ av verksamhet, en viss typ av system, eller en viss typ av utvecklingsmiljö (exempelvis ett visst programmeringsspråk). Stereotypbegreppet har lagt grunden för att olika dialekter kommer att utvecklas. Som det mesta i UML är också stereotyper öppna, det vill säga inget hindrar ett företag att utöka klassningen med rapportklasser, styrklasser och så vidare. Hur man använder UML är en öppen fråga, det är ju ett modelleringsspråk.

Enligt Ivar Jacobson är det framträdande med UML att det först och främst är baserat på användningsfall. Det är dessutom iterativt och inkrementellt. De tio första iterationerna går ut på att hitta rätt arkitektur. De därpå följande iterationerna innebär att hitta rätt programvara. Utvecklingen sker i steg och varje steg skall resultera i en körbar prototyp. Grady Booch sammanfattar problemen runt modellering i fyra punkter i en artikel i Unix Review (12/96).

Valet av modell har avgörande betydelse för vilket sätt ett problem kommer att angripas på och vilken form lösningen får. Ingen enskild modell är tillräcklig; det bästa sättet att angripa ett komplicerat system är att titta på det från olika håll. Alla modeller kan uttryckas på olika abstraktionsnivåer. De modeller som fungerar bäst är de som har den bästa verklighetsförankringen. I och med UML, the Unified Modeling Language, ska dessa problem vara lösta.

Verktygsstöd

I och med att definitionen av UML släpptes relativt nyligen, så har ännu inte verktygstillverkarna hunnit med att implementera fullt stöd för språket. Många CASE-verktyg har idag stöd för UML och genom att använda CASE-verktyg så fås maskinstöd för att enkelt skapa modeller i UML, navigera och söka i komplexa system, dokumentera modeller samt även mer avancerat stöd såsom kodgenerering, versionshantering och spårning av krav i system.

3 Företagen och deras metoder

3.1 Systemutvecklingsprocessen på Binomen

Binomen är ett relativt litet företag (19 anställda) som arbetar med administrativa system dvs. anpassningar av ekonomisystemet Pyramid och teknisk systemutveckling (på Ericsson Microwave och Volvo) De arbetar även med RCC (Rational Competence Center) vilket innebär att de hjälper kunderna med process, metod och verktygskunnande. Binomen arbetar alltid objektorienterat. Endast då kunden kräver det används strukturerad programmering, men utvecklingsarbetet sköts tills största delen ändå objektorienterat.

Binomen använder sig av en egen utvecklad systemutvecklingsmodell, Luxus®. Systemutvecklingsprocessen bygger på ett användningsfallsbaserat och objektorienterat synsätt som stöds av RUP (Rational Unified Process). Inför varje projekt skall denna metod konfigureras av projektledaren för att passa projektet (detta görs i första aktiviteten). Milstolparna skall passeras sekventiellt, även om en del faller bort eller tillkommer under konfigureringen. Tollgatesen är knutna till Binomens projektstyrningsmodell. Binomen håller även på att utveckla en ”light” version av Luxus® som skall vara till för kortare och mindre projekt och som bara skall innehålla några moduler ur Luxus®.

Luxus® är alltså sekventiell (momenten kan dock genomföras iterativt). Detta kan låta lite konstigt då Rational´s RUP (som Luxus bygger på) är iterativ och hårt trycker på fördelarna med just iterativt arbete. Detta ses dock inte som någon konflikt, utan som Jan Windahl på Binomen säger:
” Luxus® har fått ett sekventiellt utseende eftersom den är hårt knuten till vår projektstyrning. RUP:s iterativa delar framgår inte riktigt tydligt i Luxus - det är korrekt, men vi har hittills inte upplevt detta som ett problem i de mindre och medelstora projekt som vi tillämpat Luxus på. Vi brukar försöka åstadkomma iterativ parallellism, genom realisering av s.k. “återkopplande pilot-användningsfall”.

De “återkopplande pilot-användningsfallen” innebär att man väljer ut ett eller ett par representativa användningsfall som man kör i botten, inkluderande implementation och test, innan man slutför det resterande analys och designarbetet. På detta sättet så säkrar man i förväg upp att de tekniska ansatserna i arkitektur och målmiljö kommer att fungera som man har tänkt. ”

Det är även svårt att urskilja när (i vilken/vilka) aktiviteter som själva kodningen utförs. Jan Windahl förklarar detta så här:” Kodning och kodgranskning kan idag se väldigt olika ut beroende på vilka verktyg och vilken målmiljö som man väljer. Vi har inte velat låsa fast oss i någon specifik miljö och därför så är processen lite vag/flexibel på denna punkten. Men om man exempelvis väljer C++, VB, eller Java(J++) så skapar vårt huvudverktyg (Rational Rose) färdiga kodskelett för alla klasser och metoder på ett

mycket kraftfullt sätt. Det räcker oftast med att modellen granskas och bedöms den vara korrekt så blir faktiskt koden oftast därefter.”

Embryot till Luxus® föddes redan 1993 då Binomen bildades och den har därefter utvecklats och den version som används idag var färdig 1995. Eftersom Binomen har använt sig av Luxus® sedan starten 1993 kan man inte påvisa några effektiviseringar av användandet av Luxus jämfört med någon annan modell. Dock kan de se vissa effekter av Luxus® hos de företag som Binomen sålt Luxus® till. Under det allra första projektet, då modellen implementeras på ett företag, brukar både tidsmängden och kostnaderna öka, men detta skall man sedan få igen genom en ökad kompetens som i följande projekt skall spara både tid och pengar.

3.1.1 LUXUS®

Tollgate 1- Projektstart

Aktivitet- Process och verktygskonfiguration, dvs hur systemutvecklingsprocessen (Luxus ®) samt dess verktyg skall användas i projektet.
Verktyg- Word, SoDA, Clear Case
Resultat- Utvecklingsfallsbeskrivning/Development Case Description

Milstolpe 1- utvecklingsfallsbeskrivingen är klar och godkänd.

Aktivitet- Ordlista, dvs en ordlista med utmärkande begrepp och termer för projektet skapas.
Verktyg- Word
Resultat- Ordlista/Glossary

Milstolpe 2- ordlistan godkänd

Tollgate 2 a- Start av analysarbetet

Aktivitet- Verksamhetsanalys, dvs hur den verksamhet där ett tänkt informationssystem kan komma att spela en roll.
Verktyg- Word, Balance Scorecard
Resultat- Verksamhetsbeskrivning/Business Process Description

Milstolpe 3- Verksamhetsbeskrivningen klar och godkänd av kund.

Aktivitet- Kravspecifiaktion, dvs vilka speciella krav som skall ställas på systemet.
Verktyg- Rose, Requisite Pro, SoDA, UML-train
Resultat- Kravspecifikation/ Supplementary Specification

Milstolpe 4- Kravspecifikationen klar och godkänd av kund.

Aktivitet- Användningsfalls analys (Use case scenario), dvs hur aktörer (människor eller andra system i verksamheten skall interagera med det tänkta informationssystemet för att erhålla önskad funktionalitet.
Verktyg- Rose, Requisite Pro, SoDA, UML-train
Resultat- Användningsfallsbeskrivning/ Use Case Description

Milstolpe 5- Användningsfallsbeskrivningen klar och godkänd av kund.

Aktivitet- Arkitekturanalys och design, dvs indelning av systemet i logiska delsystem.

Verktyg- Rose, SoDA, Apex
Resultat- Preliminär designmodell, arkitekturbeskrivning/architectural description

Milstolpe 6- Arkitekturbeskrivningen klar och godkänd av kund.

Tollgate 2 b- Start av realisering

Aktivitet- Prototyping av användargränsnitt, dvs en prototyp av det grafiska och logiska gränssnittet mot användaren.
Verktyg- Visual Basic, Visual C++, Delphi, Smalltalk, Word
Resultat- Granskningsprotokoll från kundens genomgång av prototypen

Milstolpe 7- GUI-prototypen granskad och godkänd av kund.

Aktivitet- Testfallsanalys, dvs hur användningsfallen i användningsfallsmodellen skall testas för önskad funktionalitet skall anses vara verifierad.
Verktyg- SQA Suite, Word
Resultat- Testfallsbeskrivning/Test Case Description and procedures

Milstolpe 8- testfallsbeskrivningen klar och godkänd av kund.

Aktivitet- Användningsfall och objektdesign (systemdesignen) dvs vilka klasser som det tilltänkta informationssystemet skall kunna hantera för att uppfylla specificerade användningsfall.
Verktyg- Rose, SoDA, Apex, Word
Resultat- Desigmodell, Klassbeskrivning, Designregler

Milstolpe 9- Designmodellen klar och godkänd.
Milstolpe 10- Klassbeskrivningarna klara och godkända.
Milstolpe 11- Designreglerna klara och godkända.

Tollgate 2c- Leveransplan

Aktivitet- Releaseplanering. Denna aktivitet utförs inkrementellt i varje steg och beskriver i vilken ordning som olika användningsfall med tillhörande systempaket skall designas, implementeras och testas.
Verktyg- Word, SoDA
Resultat- Releaseplan

Milstolpe 12- Releaseplanen klar och godkänd av kund.

Aktivitet- Implementationsplanering, dvs hur olika objekt eller paket av objekt föreslås bli tekniskt implementerade (utvecklingsmiljö, fönsterdesign, klassbibliotek, databas struktur, målmiljö mm.)
Verktyg- Word
Resultat- Implementationsförslag

Milstolpe 13:n- Implementationsförslagen klara och godkända för alla i releasen ”n” berörda användningsfall.

Tollgate 3- Start av systemimplementering

Aktivitet- Implementering, dvs implementation och test av nya kodavsnitt.
Verktyg- Rose, Apex, SoDA, Purify, SQA
Resultat- Körbara moduler (källkodsfiler)

Milstolpe 14 :n- Implementering och test klar och godkänd för all kod till alla i releasen ”n” berörda användningsfall.

Aktivitet- Användningsfallstestning inklusive felrättning, dvs testning och verifiering av funktionalitet enligt testfallsbeskrivning- inklusive hjälptexter.
Verktyg- SQA Suite, Quantify, PreVue, Test Mate, SoDA
Resultat- Granskningsprotokoll – Use Case Test

Milstolpe 15:n- Användningsfallstest klar och godkänd för alla i releasen ”n” berörda användningsfall samt berörda icke-funktionella krav.

Aktivitet- Systemtest inklusive felrättning, dvs testning och validering av (del)systemet i målmiljön enligt testfallsbeskrivningen.
Verktyg- SQA Suite, Quantify, PreVue, Test Mate, SoDA
Resultat- Granskningsprotokoll- Systemtest

Milstolpe 16:n- Systemtest klar och godkänd för alla i releasen ”n” berörda användningsfall samt berörda icke-funktionella krav.

Aktivitet- Kunddokumentation, dvs all användar- och underhållsdokumentation.
Verktyg- Word
Resultat- Användar och underhållsdokumentation

Milstolpe 17:n Användar- och underhållsdokumentaion klar och godkänd för alla i releasen ”n” berörda användningsfall.

Tollgate 4a...n- Accepterad (del)leverans.

Aktivitet- Leverans, dvs kundens mottagande av delsystemet.
Verktyg- Word
Resultat- Acceptanstestprotokoll

Milstolpe 18:n- Releasen ”n” levererad med godkänd acceptans test utförd tillsammans med kund.

Aktivitet- Utbildning, dvs kundutbildning på användningen av (del)systemet.
Verktyg- Word
Resultat- Kursmaterial, kursutvärderingar

Milstolpe 19:n- kursutvärderingen sammanställd efter genomförd utbildning av användare på alla i releasen ”n” berörda användningsfall.

Tollgate 5a- Accepterad slutleverans

Aktivitet- Slutleverans, dvs Revidering av utvecklingsfallsbeskrivningen inklusive efterkalkyl.
Verktyg- Word
Resultat- Reviderad utvecklingsfallsbeskrivning samt avslutsrapport

Milstolpe 20- Intern avslutsrapport samt uppdaterad utvecklingsfallsbeskrivning.

Tollgate 5b- Projektavslut

Aktivitet- Avslut (inklusive eventuell garantitid), dvs formellt avslut av projektet.
Verktyg- Word, Balance Score Card
Resultat- Sammanställning av alla (garanti)aktiviteter samt ekonomiskt utfall för kunden baserat på vidtagna åtgärder

Milstolpe 21- Utvärdering av sammanställda garantiaktiviteter.

3.2 Systemutvecklingsprocessen på Cap Gemini

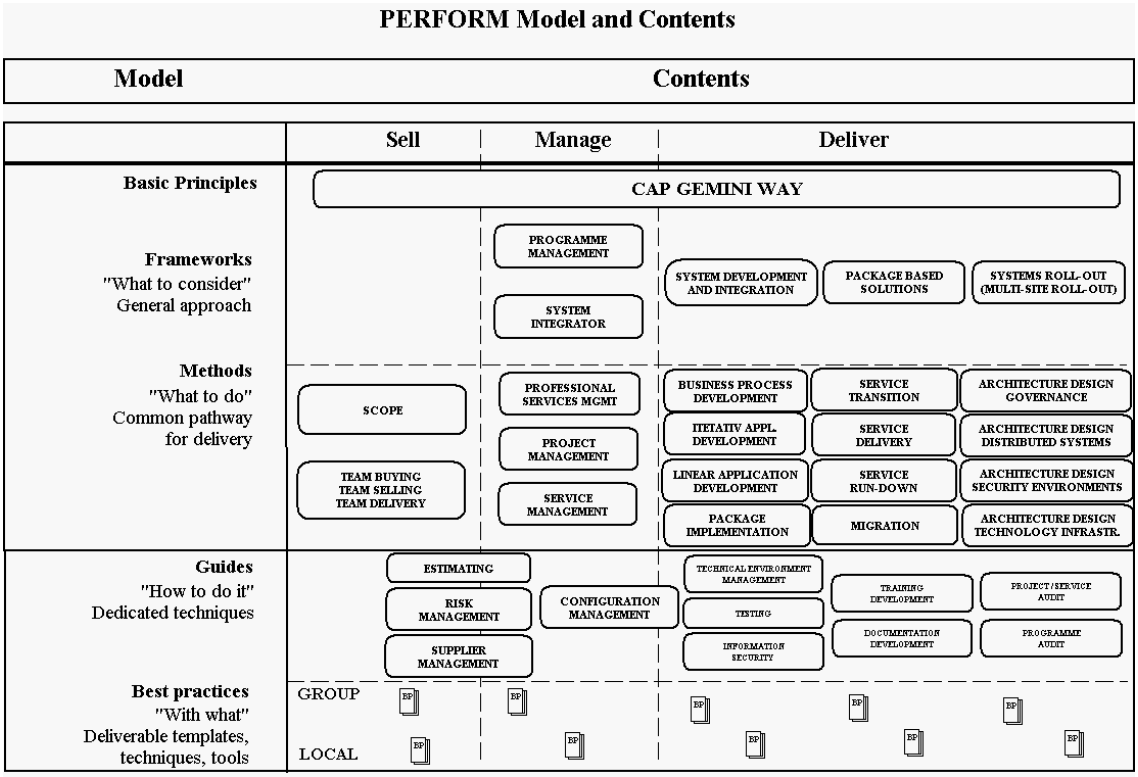
Cap Gemini är Europas och ett av världens största IT- och managementkonsultföretag. I Norden finns Cap Gemini på ett fyrtiotal orter med sammanlagt 4 300 medarbetare. Cap Gemini är certifierat enligt ISO 9001 med tillägg för TickIT.

Cap Geminis verksamhet är uppdelad i två stora huvudområden:

Projekt- och ansvarsuppdrag som utgör ca 30-40% av verksamheten och innebär att Cap Gemini själva bistår med projekledare, metoder och tar ansvar för slutresultatet.

Kompetensförstärkning som innebär att Cap Gemini säljer konsulter till kunden och att det därmed är kunden som tar ansvar för slutresultatet samt att det är kunden som beslutar om vilka metoder som skall användas.

Cap Gemini använder sig av en övergripande metod som de kallar Perform som innehåller modeller för varje slags verksamhet som Cap Gemini ägnar sig åt. Perform används slaviskt under alla aktiviteter, men eftersom Perform är uppbyggd av flera moduler kan den anpassas till varje projekt.



Figur 4

Perform är en systemutvecklingsmetod som bygger på ”best practice”, de modeller som ingår i Perform bygger på bl.a. Vattenfallsmodellen, Reflex och Logic och en iterativ modell.

Reflex och Logic är kända modeller som användes av de forna företagen Programmator och Datalogik som idag ingår i Cap Gemini.

Perform började utvecklas 1991 och då fokuserade man sig på att modellen skulle vara ett hjälpmedel för att hålla tidsramar, budget och möta kundens krav.
Viktigt var också att man skulle kunna använda modellen internationellt då Cap Gemini arbetar över hela världen, därför var ett gemensamt vokabulär en viktig aspekt.
Efter 1994, då Perform vidareutvecklades, fokuserade man sig på leveransen för att ha förmåga att leverera i tid.

Tidigare hade ansvarsprojekten en stor ”over-run” (som innebär skillnaden mellan förkalkylerad tidsåtgång och resultatet), men efter att Perform togs i bruk är det mycket små avvikelser både vad gäller budget och leveranstid. Cap Gemini använder regelbundet ”quality metrics” för att mäta kvalitén på sina produkter och har fått mycket bra betyg från sina kunder.

Perform är Cap Geminis projektstyrningsmodell och den består av en mängd olika moduler som gör att den går att anpassa till alla olika projekt som Cap Gemini arbetar med. Metoderna som används beskrivs som livscykler.

Performs livscykler är beskrivna i referensmanualer som generellt innehåller:

- Processmodell (som visar strukturen för en livscykel)
- Register för uppgifter (som beskriver de föreslagna faserna, uppgifterna och aktiviteterna)
- Register för resultat (beskriver hur man ska komma fram till resultatet)
- Register för tekniker (ger förslag på rekommenderade tekniker)
- Ordlista och bibliografi

De bitar av Perform som vi har tittat närmre på är Business Application Reference Manual och Iterative Application Development.

3.2.1 Business Application Reference Manual

Business Application Reference Manual är en del av Perform som används vid utveckling av affärssystem. Den måste användas ihop med Performs Project Management Reference Manual. Den passar för alla storlekar av affärssystemsapplikationer och uppmuntrar prototyping. Den är främst ett verktyg för projektledare som planerar utvecklingen för ett projekt, men används även av personal som deltar i utvecklingen för att visa deltagarnas ansvarsområden.

En affärssystemsapplikation har ofta följande särdrag:
Konventionell struktur, dokumentdriven, består av både automatiska och manuella systemelement (med möjlighet att länkas till andra existerande system), ingen

förutbestämd systemmjukvara, programutvecklingsspråk, användargränssnitt eller dialogprotokoll.

Man kan använda referensmanualen för affärssystem ensamt eller i kombination med andra referensmanualer för att ta fram en bild av nödvändiga aktiviteter och detta kan utföras sekventiellt eller parallellt. Den kan t.ex. föregås av en Pre-study eller Clients request, ha blivit initierat från ett systemintegrationsprojekt eller kräva användande av någon annan livscykel som t.ex. Training Development and Delivery Reference Manual eller Application Conversion Reference Manual.

Referensmanualen delar in deltagarna i nio olika roller: Projekt manager, Project quality manager, Development team, Client authority, User representative, User team, Business expert(s), Technical expert(s) och External quality authority. Ansvaret delas upp i fem olika nivåer: R = Responsible, E = Executes, P = Participates, V = Verifies, A = Approves. Rollerna och ansvarsnivåerna visas sedan i matriser i de olika skeendena.

Layout standards

Processmodell

Framsidan av processmodellen visar faser, uppgifter och aktiviteter ihop med resultat och ansvar i diagramform. Baksidan av processmodellen visar en helhetsbild av hela livscykeln och tar bara med och faser och uppgifter.

Uppgiftsregister (Directory of tasks)

Detta är ett register som är strukturerat i faser och tillhandahåller en överblick över varje fas och en beskrivning av dess uppgifter.

Teknikregister (Directory of techniques)

Detta innehåller tabeller som beskriver olika grupper av tekniker och som utförligt beskriver hur en enskild teknik används för ett visst ändamål.

Resultatregister (Directory of deliverables)

Resultatet är ett dokument som genereras vid slutförandet av en uppgift eller aktivitet. Det kan bestå av beskrivningar, diagram, listor och tablåer.

Anpassning

En livscykel anpassas efter det arbete som skall göras och det kan innebära att man tar bort en hel fas, kombinerar olika faser, aktiviteter och uppgifter, lägger till eller tar bort uppgifter och aktiviteter, upprepar faser eller uppgifter (t.ex. för sub-system) sträcker uppgifter och resultatdokument över fasgränserna.

Dessa beslut skall tas av projektledaren när planen för hela projektet läggs upp. Alla väsentliga anpassningsbeslut måste justeras i projektkvalitetsplanen (PQP) som är ett aktivt kontrolldokument där projektledaren sätter upp ramverket för projektet. PQP:n måste beskriva begränsningar och mål för både projekt och system, den globala projektplanen och nyckeldokumenten, projektets organisation och ansvarsområden, utvecklingsprocessen (faser, uppgifter och aktiviteter), projektledningsprocessen och kontrollprocessen.

Verifikation och validering

Verifikation är en process där man kontrollerar och bekräftar att ett resultat har producerats professionellt eller att en process har utförts i enlighet med överenskomna standards.

Validering är den process där man bekräftar att det som är beskrivet i resultatdokumentet kommer att tillföra en adekvat lösning till systemet och lösa kundens problem.

Uppgiftsregister (Directory of tasks)

Uppgiftsregistret är indelat i fyra faser:

- Kravspecifikation (Requirements definition)
- Funktionell design (Functional design)
- Teknisk design (Technical design)
- Realisering (Build)

Kravspecifikation

Målen med denna fas är att:

- Fastställa alla målsättningar för system och organisation och möjligheter till förbättring.
- Fastställa detaljerade krav för det nya systemet.
- Modellera dessa krav grafiskt
- Försäkra sig om att inga krav är förbisedda i det nya systemet.
- Validera föregående schema, kostnads- och riskanalys.

Kravspecifikationen innehåller följande steg:

- uppdatera PQP
- definiera systemkrav
- analysera nuvarande system
- analysera nuvarande organisation
- genomgång av teknologisk bas
- producera systemkravsmodell

Funktionell design

Målen med funktionell design är att förse kunden med en komplett funktionell modell av den rekommenderade systemlösningen, förse tekniska designers med en detaljerad funktionell specifikation som bas för den tekniska designen av systemet, bekräfta systemkonfigurationskraven och för att kunna planera nästa fas (teknisk design).

Funktionell design innehåller följande steg:

- uppdatera PQP
- skapa övergripande funktionell specifikation
- skapa systemkonfiguration
- skapa detaljerad funktionell specifikation
- definiera andra systems förändringsstrategier
- definiera organisationsförändringsstrategier
- motivera funktionell design
- definiera testningsstrategi
- producera plan över datakonvertering
- producera utbildningsplan

Teknisk design

Målet med teknisk design är att ta fram alla nödvändiga detaljer för att möjliggöra för specialister att konstruera och testa det efterfrågade systemet (eller sub-systemen) i realisationsfasen.

Teknisk design innehåller följande steg:

- uppdatera PQP
- skapa detaljerad teknisk specifikation
- utveckla implementationsplan
- planera and specificera systemtest
- specificera modifieringar till andra system
- specificera manuella procedurer
- motivera teknisk design
- planera and specificera godkännandetest
- specificera verktyg för datakonvertering
- specificera utbildning

Realisering

Målet med realisering är att konstruera operationella systemprocesser, utföra tester på dessa, modifiera andra system (där så är nödvändigt), utveckla konverteringsverktyg och

utbildningsmaterial och utveckla nödvändiga procedurer för att hantera den fysiska implementeringen av systemet.
Realisering innehåller följande steg:

- uppdatera PQP
- planera och specificera länktest
- skapa systemmanualer

I varje steg av de fyra faserna (kravspecifikation, funktionell design, teknisk design och realisering) beskrivs vad man ska ha med sig in i varje nytt steg (vilka dokument) och vad som skall produceras. Varje steg delas in i ett antal aktiviteter som skall utföras och varje steg valideras och verifieras.

3.2.2 IAD (Iterative Application Development)

Cap Gemini har investerat i och utvecklat en ny systemutvecklingsansats vars mål är att uppnå en högre grad av tillfredställda kunder genom:

- att möta kundens förväntningar
- få snabbare leveranstid
- att minska utvecklingskostnaderna

Denna nya systemutvecklingsansats är designad runt en IAD-livscykel.
IAD baseras på följande tekniska koncept:

- prototyping
- arkitektur design
- GUI design
- Återanvändning

Objektorientering kan också användas som en systemutvecklingsteknik.
Livscykeln består av ett fåtal faser som exekveras på ett iterativt sätt (se Figur 4:). Vid varje iteration genereras (och i vissa fall implementeras) en ’pilot’. Erfarenheterna från föregående pilot influerar utvecklingen av efterföljande piloter.

Följande ansatser används vid iteration beroende på situationen:

Evolutionary strategy

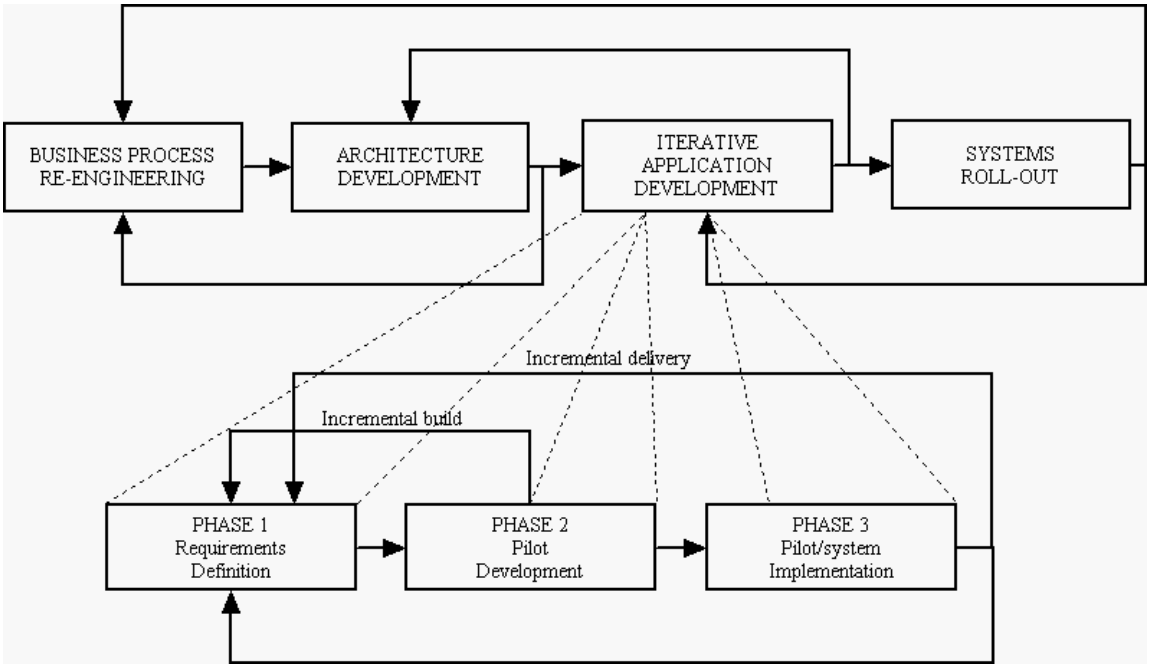
I denna strategi inkluderas alla tre faserna i varje iteration. Denna strategi reflekterar huvudkonceptet som ligger bakom en iterativ utveckling som kan summeras: *att analysera lite, designa lite, koda/testa lite, implementera lite och därefter upprepa processen.*

Incremental Delivery

I denna ansats är alla krav fullt specificerade från början och iteration används för att bygga och leverera systemet inkrementellt genom att man fokuserar på de krav som kommer att ge kunden de snabbaste fördelarna.

Incremental Build

I denna ansats använder man iteration på de två första faserna av livscykeln. Detta innebär fördelar i utvecklingsprocessen av de piloter som genererats under utvecklingen, men inget installeras förrän hela systemet är färdigt.



Figur 5

Management techniques

IAD livscykeln kräver vissa ledningstekniker:

Joint application development

Tillämpningen av workshops där användare och utvecklare tillsammans kommer överens om krav och specifikationer är typiska. Workshops leder till snabbare utvecklingscykel, realistiska specifikationer, färre dokument vid varje milstolpe och färre formella valideringsaktiviteter. Olika typer av workshops används genom IAD-livscykeln för att uppmuntra team-work och för att underlätta beslutsfattande i realtid:

- joint development strategy
- joint requirements definition
- join pilot design

- joint review and testing
- joint feedback collection
- joint acceptance

Under workshops är det uppenbarligen viktigt att ha de mest effektiva deltagarna.

A-teams

Ett A-team består av ansvarsfulla, utbildade och duktiga utvecklare med en hög grad av samarbetskänsla som tillsammans har kunskaperna som behövs för att analysera, designa, bygga och implementera systemet.

Beroende på storleken på projektet kan det finnas flera A-team som arbetar parallellt.

Time-boxing

Time-boxing är gensvaret på behovet av snabb systemutveckling och härmed kan ett leveransdatum sättas.
Time-boxing kan också utföras inom utvecklingsprocessen för att färdigställa valda delar av systemet inom en given tidsram.

Det är viktigt att användarens förväntningar är klargjorda för varje time-box. Användarna måste förstå att snabb leverans och stegvis leverans av systemet innebär begränsad funktionalitet i delrelease.

Development techniques

IAD ansatsen uppmuntrar användandet av olika utvecklingstekniker som är relevanta för storleken och stilen på ett projekt.

Prototyping

Under workshops och utvecklingsuppgifter används prototypingverktyg för att bygga olika prototyper. Vissa av dessa prototyper kan modellera det externa beteendet hos systemet likväl som den interna arkitekturen.

Det finns tre olika typer av prototyper:

- exploratory prototyping
- evolutionary prototyping
- operational prototyping

Architecture design

IAD livscykeln förespråkar en öppet och strukturerad systemarkitektur när det gäller systemutveckling. En sådan arkitektur tillför flexibilitet som är ett ’måste’ i dagens miljöer.

Denna flexibilitet fås genom:

- portabilitet
- utbyggbarhet
- testbarhet

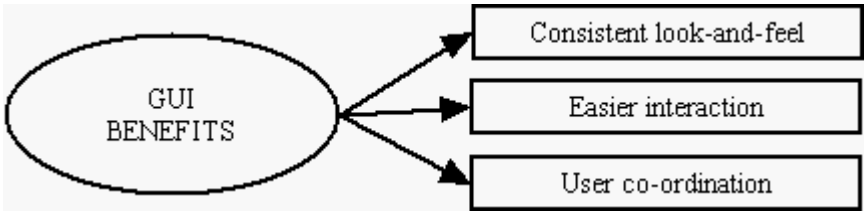
Arkitekturen bör vara så pass flexibel att den är oberoende av plattformar och nätverk. En väldefinierad arkitektur tillför också en bra bas för återanvändning och underhåll.

Graphical user interface(GUI) design

GUI handlar inte bara om ’kosmetik’ utan effektiv GUI design är viktigt då den tillför

- konsistens i utseendet på applikationerna
- lättare och mer produktiv användarsamverkan
- användarkoordination

Inom IAD livscykeln används GUI design och prototypingverktyg under workshops för att specificera och validera användarsamverkan.



Figur 6

Object orientation

IAD-livscykeln kan appliceras med hjälp av objektorienterad teknik. Objektorientering överensstämmer med mänskliga koncept genom att den representerar strukturen, dynamiken och semantiken i den verkliga världen genom abstrakta entiteter. En fördel med objektorientering är att den klarar av komplexitet och förändringar och uppmuntrar återanvändning.

Den primära enheten i objektorienterade system är ’klassen’ som representerar den gemensamma strukturen och beteendet hos en grupp objekt. Varje objektorienterat system kan därför ses som en samling av objekt som infriar vissa skyldigheter genom aktivt samarbete med andra objekt för att nå önskad funktionalitet.

Målsättningen med objektorienterad strukturering är att klara av komplexiteten i systemet genom att organisera klasser av objekt i arvshierarkier för att utforska deras likheter samtidigt som man separerar deras ansvarsområden.

Andra viktiga objektorienteringskoncept är:

- inkapsling
- polymorfism
- meddelande överföring
- dynamiska bindningar

Tack vare dessa principer är objektorienterad teknik passande för iterativ utveckling genom att den förenklar uppdelandet av problem i mindre bitar, underlättar designen av arkitekturen och ökar produktiviteten.

Reuse

Återanvändning ses ofta som nyckeln till produktivitet (genom att man kan leverera system med mindre arbete, färre fel och mindre testning).
Alla delar av en lösning bör vara återanvändningsbara, även sakkunskap.

Korttidsåteranvändning kan förbättra:

- effektiviteten
- pålitligheten

Återanvändning uppmuntrar team-work eftersom den kräver en gemensam ’kultur’ och kunskapen att bygga vidare på någon annans arbete.

Tools

Det finns en hel del verktyg tillgängliga idag för att stödja dessa tekniker och för att ta hand om komplexiteten. Bland dessa verktyg hittas:

- 3GL- och 4GL-språk
- databasmiljöer
- GUI verktyg
- Prototypingverktyg
- Objektorienterade verktyg och bibliotek

Dessa verktyg gör det möjligt att samordna utbildning och optimal återanvändning.

Method components

PERFORMs referensmanual för IAD är en modul i PERFORM. Den beskriver en metod (i form av en livscykel) som tillhandahåller en ansats för strukturering av och hur man leder projekt.

Denna ansats tillhandahåller en föreslagen struktur och en serie av de mest troliga faser, uppgifter och aktiviteter som krävs för att skapa de produkter som förväntas av utvecklingsprocessen (dokument, prototyper osv.) och till sist den färdiga produkten.

PERFORMs livscykler är beskrivna i referensmanualer som generellt innehåller:

- en processmodell (som visar livscykeln i bild)
- ett register för uppgifter (beskriver hur man kan bryta ner faser, uppgifter och aktiviteter)
- ett register för dokument (beskriver hur man bör producera dokumenten)
- ett register över tekniker (ger råd om vilka tekniker man bör använda)

Strukturen i referensmanualen för IAD är väldigt lik strukturen ovan, men det finns några skillnader p.g.a. följande:

- ansatsen som beskrivs är väldigt ny och behöver därför några specifika detaljer i dess presentation för att tillhandahålla en klar uppfattning för det underliggande konceptet
- manualen är en tidig utgåva som kommer att bli förbättrad med tiden, därför kan vissa delar vara oklara eller sakna vissa detaljer

Process model

IADs processmodell presenteras inte för sig själv utan är inbakad i IAD manualen. Processmodellen visar faserna och uppgifterna i diagramform tillsammans med de viktigaste iterationerna och de centrala workshops som bör hållas.

IAD livscykeln kan användas på två olika situationer beroende på den huvudsakliga utvecklingstekniken som används (antingen konventionella tekniker eller objektorienterade tekniker).

Directory of tasks

Uppgiftsregistret är strukturerat efter faser och ger en överblick över varje fas och tillhandahåller en beskrivning av motsvarande uppgift.

Överblicken över fasen visar fasens huvudsakliga mål, inledning, logiska grund och de centrala dokumenten. Den innehåller också dependency diagram och tillhandahåller råd om hur man skall använda sig av tuning och iteration inom fasen.

Varje uppgiftsbeskrivning innehåller följande avsnitt:

- en uppgiftsöverblick
- iterations riktlinjer
- tuning riktlinjer
- råd och kommentarer
- återanvändnings riktlinjer
- en verifikations- och valideringschecklista
- inputs och outputs
- referenser till relevanta standards och tekniker
- en aktivitetslista

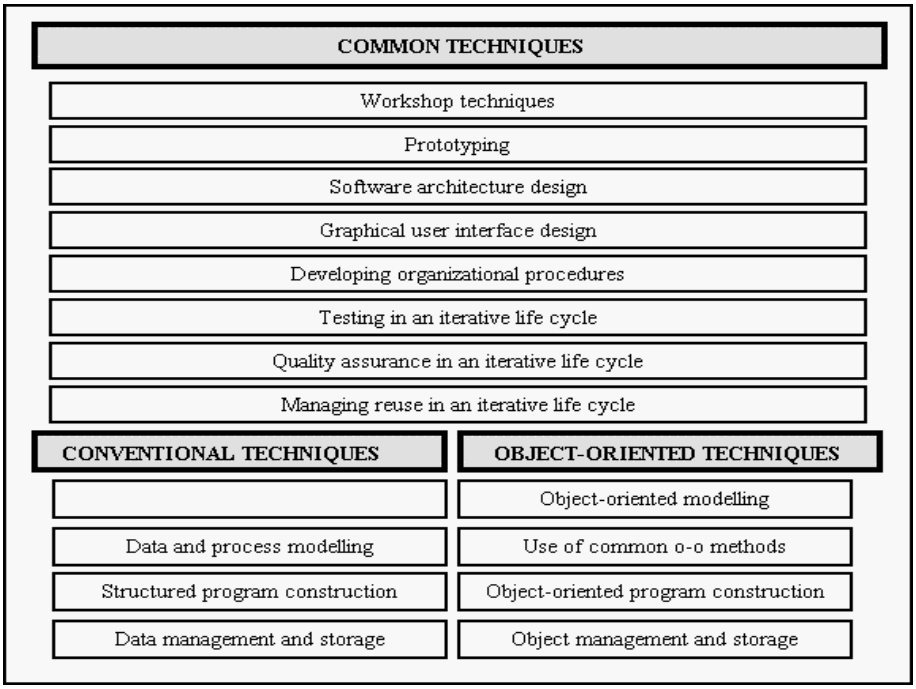
Directory of techniques

Denna presenterar en kort beskrivning om vilka tekniker som kan användas under iterativ systemutveckling.

Den innehåller tre grupper av tekniker:

- vanliga tekniker (vilkas användande är oberoende av objektorientering)
- konventionella tekniker (relaterat till systemmodellering)
- objektorienterade tekniker (ersätter konventionella modelleringstekniker när det passar i projektet)

Tabellen nedan visar listan av tekniker och till vilken grupp de hör.



Figur 7

Directory of deliverables

Resultatet är ett dokument som genereras vid slutförandet av en uppgift eller aktivitet. Det kan bestå av beskrivningar, diagram, listor och tablåer.

Andra egenskaper

Iteration och anpassning (tuning)

På grund av IAD livscykelns cykliska natur kan vissa uppgifter upprepas under den första iterationen, men sedan inte alls under de efterföljande iterationerna (och vice versa). Anpassning av en livscykel går ut på att anpassa fasernas, uppgifternas och aktiviteternas ramverk efter det arbete som skall göras. I IAD livscykeln kan det göras på följande sätt:

- flytta på fasgränser
- repetera faser eller uppgifter
- tillämpa parallell utveckling där det passar eller behövs
- utöka uppgifter och dokument över fasgränserna
- kombinera uppgifter och dokument
- ta bort eller lägga till uppgifter och aktiviteter
- flytta eller lägga till valideringar och granskningar till risker associerade med utvecklingen

Vissa av dessa beslut bör tas tillsammans under en utvecklingsstrategi workshop andra beslut tas av projektledaren då man skapar delar av den detaljerade arbetsplanen för en specifik fas. Alla signifikanta anpassningsbeslut bör beskrivas i PQP:n (project quality plan)

Verifikation och validering

Verifikation är en process där man kontrollerar och bekräftar att ett resultat har producerats professionellt eller att en process har utförts i enlighet med överenskomna standards. Denna process kan utföras manuellt eller med hjälp av automatik beroende på de verktyg som används.

Validering är den process där man bekräftar att det som är beskrivet i resultatdokumentet kommer att tillföra en adekvat lösning till systemet och lösa kundens problem. Speciella mätningar kan göras för att ge effektiv validering som t.ex. att hålla workshops och använda olika sorters prototyper.

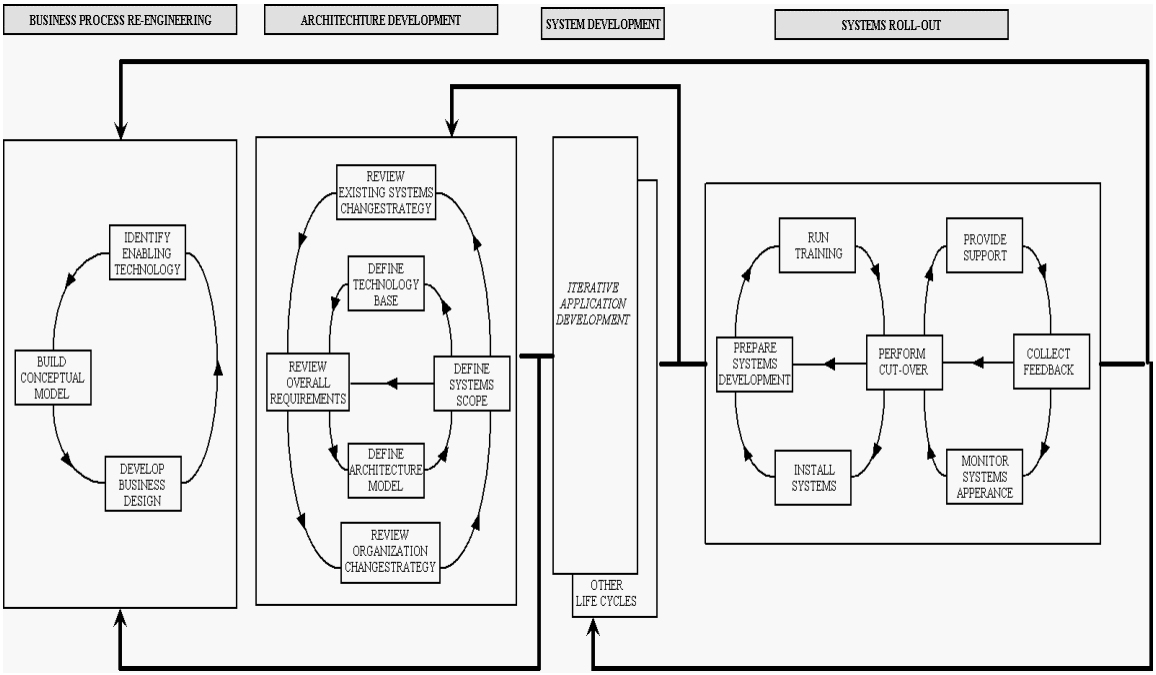
Återanvändning

Återanvändning begränsas inte bara till mjukvarukomponenter utan det kan även omfatta alla designdelar som används under hela livscykeln. Därför innehåller varje uppgiftsbeskrivning en separat sektion för återanvändning.

Återanvändningsaktiviteterna innebär:

- att identifiera möjligheterna att hitta likheter
- bearbeta dessa möjligheter

Användandet av de nya komponenterna leder till att man får uppslag till ny a förändringar på designdelarna.



Figur 8

Business Process Re-engineering

BPR (Business process re-engineering) är en iterativ process där man använder sig av mycket erfarna företagsrepresentantanter som assisteras av medhjälpare och teknologiska visionärer. BPR innebär nytänkande och nydesign av ett helt företag, processer, ledning, jobb, organisationens struktur och värderingar inkluderat.

BPR ses som en metod för att:

- förbättra kundservicen
- öka konkurrensmöjligheterna
- förhöja flexibiliteten i funktionerna
- minska kostnaderna på IT system

BPR utforskar potentiella vägar, expanderar företagsvisionen och utvecklar företagets processmodeller.

Som en konsekvens kommer de gamla processerna att bytas ut mot nya och detta kommer i sin tur att ha en positiv effekt på företaget och de anställda inom organisationen. I en sådan här miljö har informationsteknologi blivit ett bra hjälpmedel vid implementationen av mer effektiva företagsprocesser.

Architecture development

Den här processen har sin egen livscykel, den får sin ’näring’ från BPR och den producerar en teknisk arkitektur som underhåller de strategiska IS kraven.
Processen använder sig av output från BPR för att se över kraven så att man kan välja de informationsssystem som möter företagskraven. Sedan utforskar den de potentiella alternativen till IT arkitektur och bekräftar vilken effekt den arkitekturen kommer att ha på existerande system och företaget.

Arkitekturutvecklingen inkluderar också implementations planen för de tekniska komponenterna, underhållet, hjälpen och utbildningsbehovet.

Systems development

All systemutveckling kan innehålla en (eller flera) initiativ när man använder IAD och parallella initiativ när man använder sig av andra livscykler.

Alla Perform-systemutvecklingslivscykler kontrolleras av Perform-projektledningslivscykeln som hanterar planering, uppdelning av uppgifter, kontroll, rapportering och leverans av produkten.

Om IAD’n följer en BPR och/eller arkitekturutvecklingsvägen kommer dessa att tillföra de tekniska anvisningarna och de globala kraven för de önskade systemen/applikationerna.

Systems roll-out

Denna del är viktigare än implementeringen av en lokaliserad pilot, och kanske även hela systemet, inom ramarna för IAD utveckling.

Utvecklingen kan ha handlat om en del under en avdelning där det används specifika pilothårdvara och mjukvara eller nätverk.

När man sen accepterat denna del för implementation över hela koncernen är ’roll-outen’ en mycket stor och komplicerad spridningsprocess som kräver planering, inköp, installation, träning, acceptans, support och feedback.

När det gäller stora strategiska system kräver hårdvara, mjukvara, nätverk, support och konfiguration av alla delar en tillägnad livscykel.

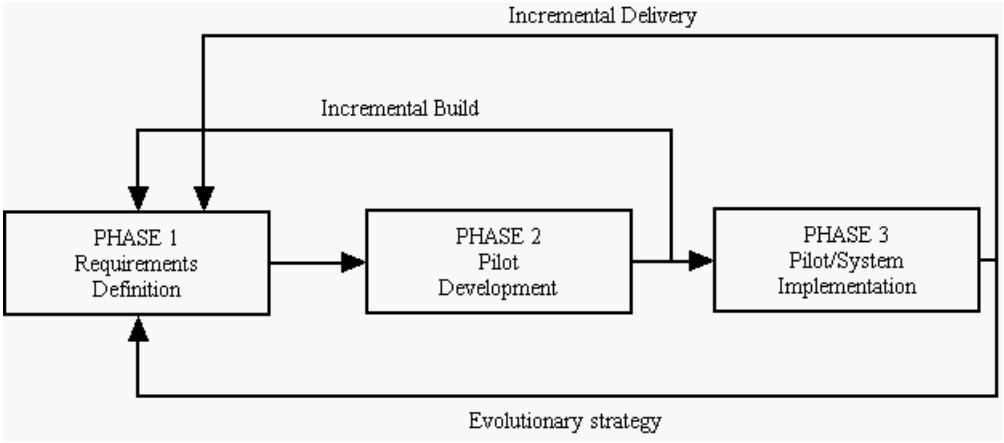
IAD faserna

Målet med IAD livscykeln är att snabbt utveckla en applikation genom multipla iterationer av följande tre faser:

- Requirements definition
- Pilot Development
- Pilot/System Implementation

Tre huvudsakliga ansatser kan användas vid iteration beroende på komplexitet eller stabilitet vad gäller kraven, eller behovet av snabb implementation av systemet:

- Evolutionary strategy
- Incremental delivery
- Incremental build



Figur 9

Vid varje iteration levereras en version av systemet som genast kan börja användas. Varje version kallas *Pilot.* Feedback på den implementerade piloten används som input i nästa livscykeliteration. Den kommer att påverka specifikationen och utvecklingen av efterföljande piloter.

Requirements definition

Målet med kravdefinitionsfasen är att analysera målen och restriktionerna på systemet i nära samarbete mellan användare och systemets utvecklare.

Detta fås genom interaktiva *Joint Requirements Definitions* (JRD) workshops efter att man har kommit överens om den huvudsakliga utvecklingsstrategin.

När kravdefinitionsfasen är en integrerad del av varje iteration blir erfarenheterna från tidigare piloter viktig input till varje JRD workshop. Exploratory prototyping används

också för att klargöra vissa oklara krav och för att kunna skapa en första skiss på specifikationer på vissa delar av systemet.

Baserat på en lista med systemkrav, en teknisk arkitektur och en definierad teknologisk bas modelleras ett globalt koncept av systemet och en plan över iterationer ritas upp. Denna plan kallas Pilotplan.

Pilotplanen kommer hela tiden att förbättras och få fler detaljer där det beskrivs vilka krav som kommer att täckas av vilken pilot, till hur stor del och i vilken ordning. Pilotplanen används i slutet av varje större iteration för att avgöra om fler förbättringar behövs i systemet och den används som en guide till all fortsatt aktivitet i de två följande faserna i IAD'n.

Pilot development

I pilotutvecklingsfasen designas, utvecklas och testas en komplett pilot. Användarna är med i planläggningen av piloten under *Joint Pilot Design* (JPD) workshops. Workshopkonceptet försäkrar att funktionaliteten hos piloten kan specificeras och valideras korrekt utan att det tar extra tid vad gäller milstolpar och dokument. Exploratory prototyping 'på plats' kan användas för att modellera diverse funktionella eller tekniska aspekter.

Delar av piloten kan sedan konstrueras parallellt av flera a-team för att spara tid eller för att bygga större system inom den begränsade tidsramen för en livscykeliteration. Under pilotutvecklingsfasen kan piloten byggas under flera 'under'-iterationer. Demoversioner av piloten (kallade pilotinkrement) byggs och valideras under *Joint Review och Testing* (JRT) sessioner.

Worksshopkonceptet bör korta ner valideringstiden och bearbetning av feedback. Den eliminerar också behovet av stora volymer av detaljerade specifikationer eftersom man kommer överens om och bestämmer sig för detaljerna under workshop mötena. Resultaten från JRT sessionerna kommer att bli input till ännu en JPD workshop, speciellt då en workshop inte räcker för att komma fram till alla pilot specifikationer.

Pilot/System implementation

Under Pilot/System implementationsfasen blir piloten accepterad och implementerad i organisationen. Om piloten är slutgiltig dvs. att inga fler iterationer är planerade refererar man till den som *systemet.*

Men oftast blir det fler iterationer. Under Pilot/System implementationsfasen samlar man in feedback för att användas som input till efterföljande livscykel iterationer.

De centrala principerna av IAD ansatsen är:

- det höga engagemanget från slutanvändarna under intensiva workshops genom hela livscykeln
- förmågan att visualisera och verifiera krav genom användning av avancerade utvecklingsverktyg
- den inkrementella leveransen av mjukvara som försäkrar att applikationen är byggd efter de senaste kraven
- iteration genom användning av prototyping och formella modellerings tekniker
- förmågan att klara av risker genom kontinuerlig projektgranskning och begränsning av iterationer

Det bör också tilläggas att IAD ansatsen har stark effekt på:

- management practices (iterationskontroll, riskhantering)
- technical practices (tonvikt på arkitekturdesign, ingen ’big bang’ utan evolution, snabb leverans på de viktigaste delarna av systemet)
- client relationships (högre engagemang från användarna, nya avtalsenliga aspekter, behovet av tillit och engagemang)

3.3 Systemutvecklingsprocessen på Ericsson Mobile Data Design (ERV)

Ericsson Mobile Data Design utvecklar mobila datakommunikationsnät, dvs. stora tekniska projekt. Ett av de större projekten som Ericsson Mobile Data Design arbetar med just nu är GPRS, en efterföljare till GSM där man kan 'skicka paket' av data via mobil. Ericsson Mobile Data Design har en modell som heter Darwin som är en övergripande processmodell.

Darwin bygger på PROPS som är en projektstyrningsmodell på koncernnivå. Darwin är en vattenfallsmodell, men med vissa iterativa moment.

Darwin är 'minsta gemensamma nämnaren' för hur arbetet skall göras, man kan lägga till eller hoppa över faser dvs. adaptera modellen efter projektets inriktning. Det är dock svårt att få alla att använda sig av Darwin då systemutveckling är en kreativ process och formalisering av denna kan inskränka på utvecklarnas konstnärliga arbete.

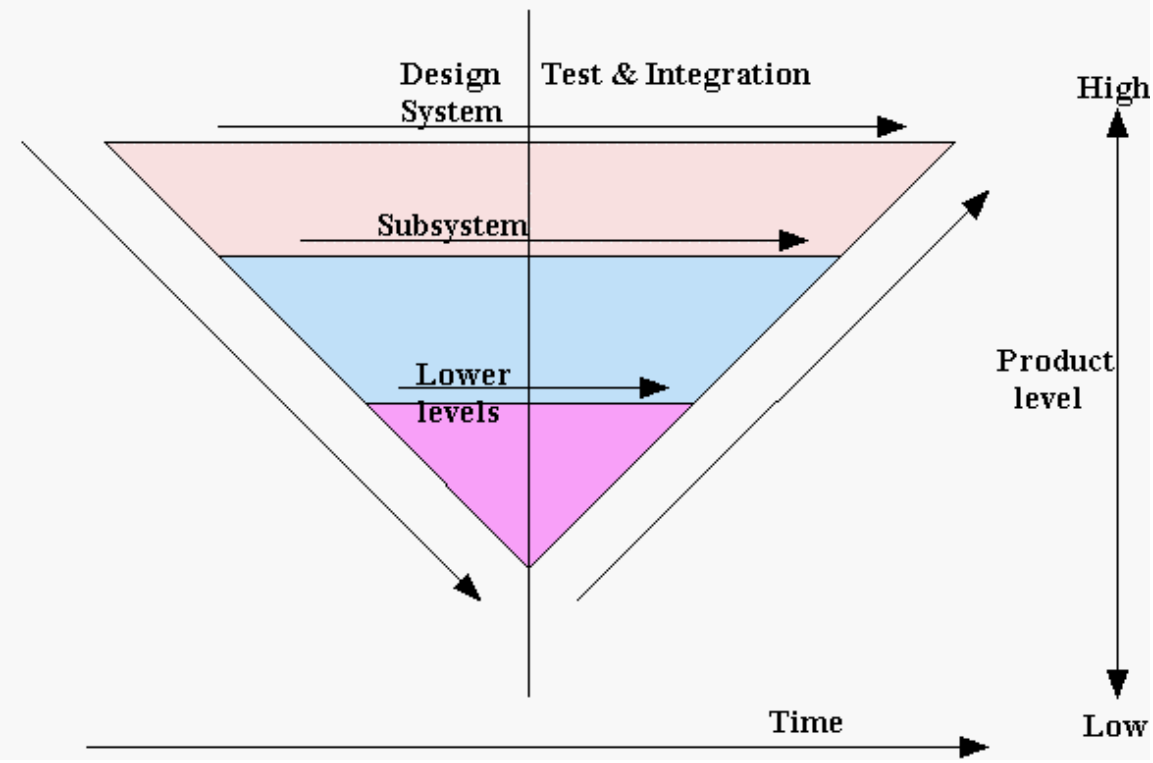
Modellen har använts sedan 1997. Innan detta arbetade man mycket ad hoc.
För att skapa Darwin dokumenterades redan befintliga processer steg för steg och på så vis skapades utvecklingsmodellen. Den har allt sedan dess, så som namnet antyder, utvecklats och omarbetats utifrån de erfarenheter man dragit av användandet. Man har alltså tagit bort de delar som var mindre bra och tagit fasta på de delar som visat sig vara effektiva.

Då vi frågade Anna Börjesson på ERV om metoden har effektiviserat arbetet på företaget svarade hon: "Ja, det har den för nu vet man vad man håller på med".

Att det effektiviserat arbetet att använda sig av Darwin ser man på att informationen inom projekten blivit mera lättillgänglig. Det har också blivit lättare för nyanställda att komma in i hur man arbetar med projekt och detta är en stor fördel med tanke på att Ericsson Mobile Data Design har vuxit kraftigt sedan 1995. Darwin har även givit möjlighet att hantera Change Request, ändring i kravspecifikationen, på ett mera effektivt sätt.

3.3.1 Darwin

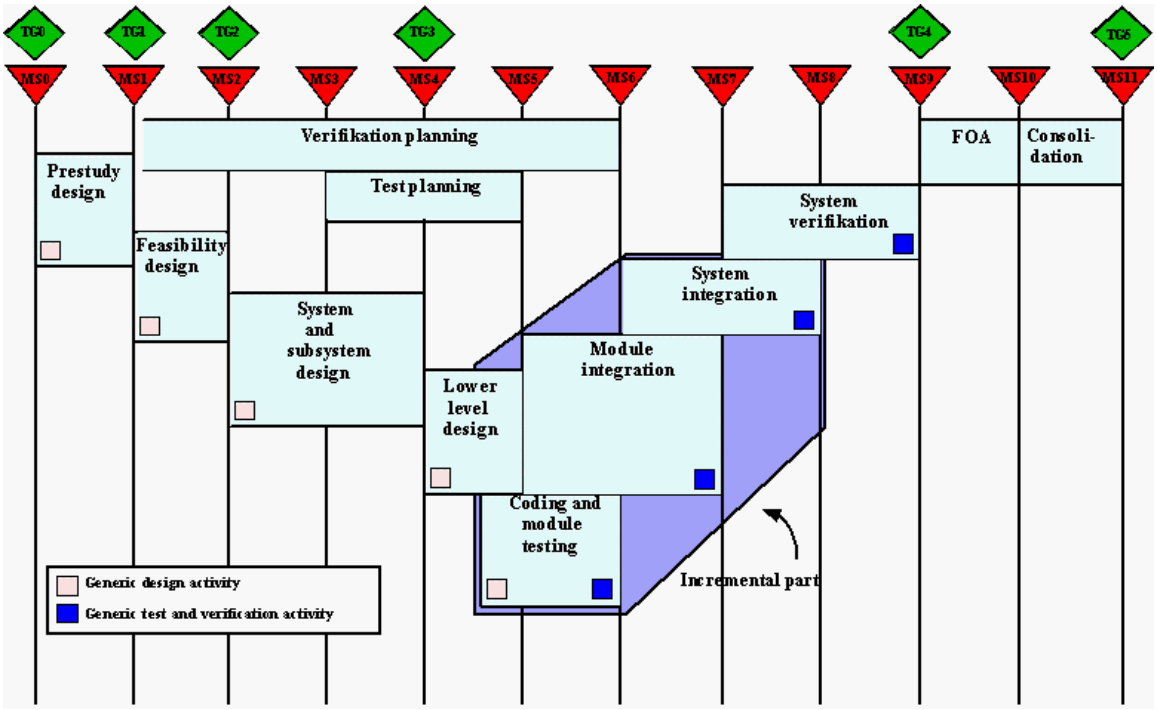
Nedan följer en beskrivning av Darwin. Vissa delar av Darwin tas med som bilagor i slutet av arbetet då vi inte velat lägga någon större tyngd på att mer utförligt förklara dessa i arbetet.
Grundprincipen i Darwin är "V-modellen" (Figur 10). Den beskriver arbetsflödet i relation till tid och produktens abstraktionsnivå. Utvecklingsarbetet börjar i det översta vänstra hörnet, med systemdesignen, fortsätter nerför V-t till lägre design nivåer och slutligen kodning allra längst ned. På vägen uppför den högra sidan görs integration, testning och verifiering av systemet. V-modellen har sedan specifierats till denna sk."Process Map". (Figur 11)



Figur 10

”Tollgates” (de gröna fyrkanterna) är knutna till PROPS som är hela Ericssons projektstyrningsmetod. Enligt den så delar man in projekten i fyra olika faser; Prestudy (förstudie), Feasibility study (genomförbarhet), Execution (realiseringsfasen) och Conclusion (projektavslut). (Se Figur 11)

Prestudy är den första fasen då projektets omfattning och avgränsningar utarbetas och klargörs. Under Feasibility study klargörs huruvida målet kan nås, och hur Execution skall genomföras för att nå målet. Projektet planeras, resurser säkras och möjliga tekniska implementationer diskuteras och föreslås. Under Execution tar själva utvecklingsarbete plats. Execution delas in i design, implementation och testning. Till Executionsfasen räknas också First Office of Application, som är det första testet hos kunden och där systemet till sist slutligt verifieras.



Figur 11

Activities

Aktiviteterna, rutorna i Figur 11, är själva byggstenarna i det som Darwin utgör. Varje aktivitet består av ett antal handlingar. Under utvecklingsprocessen sker vanligtvis två typer av aktiviteter- design och test. Därför följer de flesta aktiviteterna en generell mall, vilket kommer att förklaras nedan. Tanken är då att designen utförs som flera iterationer runt en specifik modell. Varje design aktivitet är alltså en detaljerad beskrivning av iterationens specifika händelser.

The iterative models

Design

Designen sker i flera steg, men designern måste ändå se helheten i varje steg. I Darwin beskrivs detta som ett antal olika områden som alltid måste beaktas. Dock fokuseras arbetet i varje designaktivitet huvudsakligen bara på ett eller två av dessa områden. Fokuseringen ändras då allteftersom projektet fortskrider nedför ”V-modellen”.

Test and Verification

Situationen är liknande för test och verifikations aktiviteterna, men de är ordnade på ett något annorlunda sätt. Varje test och verifikations aktivitet är en mer separat enhet, med ett mera traditionellt ”top-down” tillvägagångssätt.

Generell mall för design

Den generella mallen för design tillämpas på följande aktiviteter:

- Prestudy design
- Feasability design
- System and subsystem design
- Lower level design
- Coding (del av Coding and module testing)

Det generella arbetsflödet innefattar åtta olika designaspekter (visas nedan). Dessa specificeras vidare i varje designaktivitet i Darwin.

Mallen

Den generella mallen för design är en komplett designcykel. Den innefattar alla aspekter som måste beaktas när man designar ett mjukvarusystem. Varje designaktivitet är som sagt en iteration av denna mall, där skillnaden är var man lägger fokus och graden av abstraktion

Aspekter	Förklaring
Function	Identifiera och beskriv vilka uppgifter systemet skall utföra och tillhandahålla, och systemets egenskaper.
Distribution	Definiera hur funktionerna distribueras i systemimplementationen.
Implementation Design	Definiera hur varje del av systemet skall implementeras.
Prototyping	Undersök vissa möjliga implementationer (kan också göras genom simuleringar).
Coding	Definiera regler och riktlinjer för hur den verkliga implementationsaktiviteten skall genomföras. Skriv till slut koden.

Testing	Se till att de implementerade delarna är designade på ett sådant sätt att testning av systemet blir lätt och effektivt genomförbart.
Productification	Identifiera och klara av de delar av systemet som innehåller status och andra delar av styrande information.
Industrialisation	Klargör hur de olika delarna i systemet skall behandlas för att lämna designstadiet.

Designdokumentations-förklaringar finns bifogade i bilaga två.

Generell mall för Test and Verification

Den generella mallen för test och verifikation tillämpas på följande aktiviteter:
- Module testing (del av Coding and module testing)
- Module integration
- System integration
- System verification

Det generella arbetsflödet innefattar fem olika delar (visas nedan). Dessa specificeras vidare i varje test och verifikations aktivitet i Darwin.

Det generella arbetsflödet:
- Planning
- Specification
- Execution
- Recording
- Checking for completion

Det normala arbetsflödet är uppifrån och ned. Denna modell tillämpas på alla test och verifikations aktiviteter, skillnaden är vilka *typer* av test och verifikation som genomförs.

Test and Verification tabell

Denna tabell visar ett antal av de olika aspekter som måste beaktas på den högra sidan av ”V-modellen”. För varje aspekt visas i vilken test och verifikations aktivitet aspekten skall beaktas och i vilka den inte bör beaktas.Den generella regeln är att alla aspekter skall beaktas någon gång under projektet.

Förkortningar: M=mandatory, detta måste göras i denna aktivitet!
O= optional, detta kan göras i denna aktivitet, men det kan också göras i någon annan del.
N= not, detta skall INTE göras i denna aktivitet! Det skall göras antingen innan eller efteråt.

The test & verification table

Test & verification aspect	Activity			
	MT Coding & module test	FT Module integration	SI System integration	SV System verification
Boundary value testing	M	N	N	N
User interface testing	M	T.B.D.	T.B.D.	T.B.D.
Error case testing (negative testing)	M	O	M	O
Functional tests	M	M	M	N
Integration	N	M	M	N
Integration testing (interface testing)	N	M	M	N
Performance testing	O	M	M	N
Legal approval (certification of radio characteristics, EMC, etc.)	-	-	M	N
Testing backwards compatibility (when applicable)	O	M	M	O
Recovery testing		O	M	M
Customer Product Information (operational manuals for the customer)	N	N	M	M
Performance verification	N	N	N	M
Use case testing	-	-	O	M
Stress testing	N	O	O	M
Installation testing	-	-	O	M
Regression verification (note that there are different test cases on each test level)	M	M	M	M

Figur 12

Aktiviteterna

Prestudy design
Syftet med Prestudy design är att sätta projektets mål i tekniska termer. Syftet är alltså att skrapa på ytan av en möjlig impementation, att utvärdera om huruvida kraven är rimliga. Denna aktivitet sätter fokus på behovet av att både designers och testare är med tidigt i projektet Testbarhet och underhållsmöjligheter måste beaktas. Denna fas skall slutföras vid Milstolpe 1 (MS1).

Aktivitet:

- Specificera alla krav tillsammans med beställaren. Specificera även ERV´s interna krav. Se till att ursprunget till varje krav dokumenteras. När man senare kan bli tvungen att prioritera kraven eller slopa funktionalitet är denna information oerhört värdefull.
- Använd ”Prestudy design” tabellen (se bilaga tre) som en guide för vad som skall göras och vilka dokument som skall produceras.
- Välj ett projektarkiv, och förbered det för användning.

Input:

- Uppdraget, kravlistor eller liknande dokument från beställaren, vilka sätter målet för projektet.
- Om lämpligt, existerande dokumentation och produkter.

Output:

- Kravspecifikation
- Implementationsskisser
- Tekniska rapporter
- Preliminär funktionell produktstruktur

Feasibility design

Syftet med denna fas är att identifiera vad som behövs för att uppnå de krav som specificeras i kravspecifikationen. Testbarhet och underhållsmöjligheter måste beaktas. Denna fas skall slutföras vid Milstolpe 2 (MS1).

Aktivitet

- Använd ”Feasibility design information” tabellen (se bilaga fyra) som en guide för vad som skall göras och vilka dokument som skall produceras.
- Utför riskanalys.

Input

- Kravspecifikation
- Om lämpligt, existerande dokumentation och produkter.

- Implementationsförslag
- Tekniska rapporter
- Preliminär funktionell produktsstruktur

Output

- Uppdaterad kravspecifikation
- Implementationsförslag
- Projektspecifikation
- Dokumentlista
- Tekniska rapporter
- Funktionell produktstruktur
- Preliminär designstruktur
- Preliminär leverensplan
- Strukturspecifikation

System and subsystem design

Målet med denna aktivitet är att specificera innehållet i den funktionella strukturen på en detaljerad nivå. Testbarhet och underhållsmöjligheter måste beaktas. Denna fas skall slutföras vid Milstolpe 4 (MS4) med ett delresultat vid Milstolpe 3 (MS3).

Aktivitet

- Använd ”System och subsystem design information” tabellen (se bilaga fem) som en guide för vad som skall göras och vilka dokument som skall produceras.

Input

- Kravspecifikation
- Implementationsförslag
- Funktionell produktstruktur
- Om lämpligt, existerande dokumentation och produkter.
- Tekniska rapporter

Output

- Funktionalitetsbeskrivning
- Funktionsbeskrivning
- Systemets uppbyggnads plan
- Design specifikationer (Rose[4] modeller om möjligt)
- Gränssnittsbeskrivningar (Rose modeller om möjligt)
- Produktstruktur
- Uppdaterad strukturspecifikation
- Designregler

[4] Rose- Modelleringsverktyg från Rational

Lower level design

Syftet med denna fas är att analysera all funktionalitet och krav på systemets olika delar, och bestämma hur de skall implementeras. Testbarhet och underhållsmöjligheter måste beaktas. Detta görs både innan den inkrementella fasen börjar, och inom varje inkrement. Slutförs vid Milstolpe 5 (MS5).

Aktiviteter

- Använd ”Lower level design information” tabellen (se bilaga sex) som en guide för vad som skall göras och vilka dokument som skall produceras.

Input

- Kravspecifikation
- Produktstruktur
- Funktionsbeskrivning
- Om möjligt, Rose-modeller
- Design specifikationer och gränssnittsbeskrivningar
- Om möjligt, existerande produkter och dokumentation

Output

- Funktionsspecifikationer på subsystemnivå
- Information i Rose-modeller eller designspecifikationer
- Källkodsskelett
- Skelett för gränssnittsfiler
- Uppdaterad strukturspecifikation
- Information för kodgenerering
- Ordnade data
- Ordnad information

Verification planning

Denna fas innebär att man planerar de tester och verifikationer som skall göras på systemnivå. Detta innefattar aktiviteterna System Integration och System Verifikation. Fasen avslutas vid Milstolpe 6 (MS6), med delresultat vid Milstolpe 2 (MS2) och 4 (MS4).

Aktiviteter

- Upprätta en strategi för verifieringen av systemet
- Ta reda på hur noggrant varje produkt skall testas och vilka testfall som täcker dessa krav.
- Säkerställ att testfallen täcker alla aspekter i testtabellen
- Undersök behoven för nya eller förbättrade testverktyg
- Anskaffa testmiljön

Input

- Kravspecifikationen
- Leveransplan
- Produktstrukturer

Output

- Verifikationsstrategi (MS2)
- Verifikationsplan (MS4)
- Preliminära specifikationer för systemintegration och verifikationsaktiviteter
- Laborations beskrivning (Preliminärt vid MS2, godkänd vid MS4)

Test planning

Denna fas innefattar planering av testningsaktiviteterna från implementation upp till subsystemnivå. Detta inkluderar Module Testing och Module Integration.
Avslutas vid Milstolpe 5 (MS5).

Aktivitet

- Planera testningsaktiviteterna, dvs hur testnings- och integrationsarbetet skall genomföras av testningspersonalen.
- Ta reda på hur noggrant varje produkt skall testas och vilka testfall som täcker dessa krav.
- Säkerställa att testfallen täcker alla aspekter i testtabellen
- Planera för vilken ordning de olika modulerna skall integreras och vilka resurser som behövs för detta.
- Undersök behoven för nya eller förbättrade testverktyg
- Anskaffa testmiljön

Input

- Kravspecifikation
- Leveransplan
- Produkt strukturer
- Verifikationsstrategi

Output

- Testplan
- Preliminära testspecifikationer, (vid denna tidpunkt skall de beskriva vilka testfall som behövs, men inte hur de skall genomföras)

Coding and Module Testing

Syftet med denna fas är att förbereda fungerande kod på ett sätt som minimerar källor till fel. Detta görs i varje inkrement och slutförs vid Milstolpe 6 (MS6). Det är också här som design och testdelen i ”V-modellen” möts. Kodning och modultestning är en och samma aktivitet, då den utförs av en och samma utvecklare, och aktiviteten itereras på ett komplext sätt.
Denna aktivitet baseras på både den generella mallen för design *och* testning.
Syftet med kodningen är att producera kod som stämmer överens med arkitekturen av systemet och designen som gjorts tidigare.
Syftet med modultestningen är att finna så många fel som möjligt i koden innan modulen integreras med andra. Modultestningen är obligatorisk. När man utformar testfallen och utför testen så skall ”Guidelines for Module Testing” användas (finns ej beskriven). Det finns inga krav på hur testfallen och testens resultat dokumenteras, bara *att* de dokumenteras. De samlade testresultaten från alla genomförda tester skall dokumenteras i en test-journal.

Roller och ansvar

Vem?	Roll / Ansvar
Vem utför testerna / kodningen?	Varje utvecklare
Vem är huvudsakligen ansvarig?	Delprojektledaren eller "teamleadern"
Vem får resultaten av testerna / kodningen?	Delprojektledaren eller "teamleadern"

Aktivitet

Ur designaspekten: Kodning

- Skapa/uppdatera koden

Ur designaspekten: Testning

- Genomför debugging och korrigeringar (informella tester).
- Bestäm kraven som modulen måste kunna uppnå för att gå igenom testningen.
- Genomför testning enligt ”ERV´s Module Testing Policy” och enligt ”Guidelines for Module Testing”. Alla moduler skall godkännas enligt de krav som bestämts ovan.
- Använd checklistan /protokollen för modul testning och kodgranskning.
- Säkerställ att testfallen täcker alla aspekter i testtabellen.

Ingångskriteria

Kodning och modul testspecifikations delen:

- Designspecifikationen för alla moduler är klar.
- Testplanen är klar.
- Kodningsregler är definierade.
- Utvecklingsmiljön är klar.

Modultestnings exekverings delen:

- Implementationen är slutförd.
- Modultestnings specifikationerna är klara. Detta kan vara dokument som beskriver testfallen, testningsmiljön etc.
- De rätta verktygen för testningen är redo.

Utgångskriteria

- All kod är skriven enligt specifikationerna.
- Alla nya/modifierade moduler är testade.
- Alla testresultat har dokumenterats.
- Alla fel skall ha hittats och korrigerats.

Input

- Designspecifikationer och gränssnittsspecifikationer
- Rose-modeller
- Preliminära testspecifikationer
- Testplan
- Verifikationsstrategi
- Laborationsbeskrivning

Output

- Testad källkod
- Testspecifikationer, dvs dokument beskrivande testfallen, testmiljön etc.
- Testjournal som beskriver testresultaten

Module Integration

Modulintegrationsfasen innebär att man sätter samman mindre moduler till större enheter. Funktionaliteten testas, samt gränssnitten och samverkan mellan modulerna.
Denna aktivitet avslutas vid Milstolpe 7 (MS7) med delresultat vid Milstolpe 6 (MS6).

Aktivitet

- Specificera testfallen enligt testtabellen
- Specificera testmiljön
- Utveckla testmiljön
- Integrera kodmodulerna
- Testa modulerna allteftersom de sätts samman
- Rapportera problem
- Gå tillbaka till adekvata aktiviteter och genomför de förändringar som behövs.

Ingångskriteria

Specifikationsdelen:

- Testplanen är klar

Exekveringsdelen:

- Testspecifikationerna är klara
- Testmiljön är klar
- Koden är modultestad och redo för integrering

Utgångskriteria

- Modulintegrationen är genomförd och dokumenterad
- Alla fel gällande interaktionen mellan modulerna och gällande subsystemens funktionalitet är hittade, åtgärdade och verifierade

Input

- Kravspecifikation
- Produktstruktur
- Funktionsspecifikation
- Designspecifikation och gränssnittsbeskrivning
- Rose-modeller
- Preliminära testspecifikationer för de berörda delarna
- Testplan
- Verifikationsstrategi
- Laborationsbeskrivning
- Modultestad kod

Output

- Testspecifikationer för varje del
- Testresultat för varje testspecifikation
- Testjournal

System Integration

Syftet med system integrationsaktiviteten är att kontrollera interaktionen mellan subsystemen och funktionaliteten. Denna aktivitet genomförs för varje inkrement. Specifikationsdelen avslutas vid Milstolpe 7 (MS7) och testningsdelen vid Milstolpe 8 (MS8).

Aktivitet

- Specificera testfall enligt testtabellen
- Specificera testmiljön
- Utveckla testmiljön
- Integrera alla subsystem till ett system
- Utför testning
- Rapportera problem
- Gå tillbaka till adekvata aktiviteter och genomför de förändringar som behövs.

Ingångskriteria

Specifieringsdelen:

- Verifikationsplanen är klar

Exekveringsdelen

- Verifikationsspecificeringarna för systemintegrationen är klar
- Modulintegrationen är klar och dokumenterad

Utgångskriteria

- Systemintegrationen är utförd och dokumenterad
- Alla fel gällande den funktionalitet som systemet skall tillhandahålla är hittade, åtgärdade och verifierade

Input

- Kravspecifikation
- Funktionsbeskrivningar
- Produktstruktur
- Preliminära verifikationsspecificeringarna för systemintegrationen
- Noggrant testade subsystem
- Verifikationsplan
- Verifikationsstrategi
- Laborationsbeskrivning

Output

- Verifikationsspecifikation för systemintegrationen
- Verifikationsresultat för varje verifikationsspecifikation
- Verifikationsjournal
- Verifikationsrapport

System Verification

Syftet med systemverifikationen är att installera, använda och underhålla hela systemet på ett sätt som man kan förvänta sig att systemet kommer att behandlas efter det är satt i bruk. Stabilitet och korrekthet, såväl som produktinformation till kunden kontrolleras. Aktiviteten avslutas vid Milstolpe 9 (MS9) med delresultat vi Milstolpe 8 (MS8).

Aktivitet

- Specificera testfall utifrån testtabellen
- Specificera testmiljön
- Utveckla testmiljön
- Utför verifikationen
- Rapportera problem
- Verifiera produktinformationen för kund, t.ex. installation och användarmanualer
- Gå tillbaka till adekvata aktiviteter och genomför de förändringar som behövs.

Ingångskriteria

Specificeringsdelen:

- Verifikationsspecifikationen är klar
- Verifikationsplanen är klar

Exekveringsdelen:

- Systemet är integrerat, med alla funktioner som skall gå vidare till First Office Activity (FOA)
- Tidigare tester är avslutade och alla fel är åtgärdade och verifierade.
- Testmiljön är redo (rätt utrustning vad gäller hårdvara och mjukvara) och valda testverktyg är redo

Utgångskriteria

- Systemverifikationen är slutförd och dokumenterad
- Alla fel gällande systemets karaktäristika är hittade, åtgärdade och verifierade

Input

- Kravspecifikation
- Produktstruktur
- Preliminära verifikationsspecifikation
- Preliminär produktinformation till kund
- Det integrerade systemet
- Verifikationsplanen
- Verifikationsstrategi
- Laborationsbeskrivning

Output

- Verifikationsspecifikation
- Verifikationsresultat
- Verifikationsjournal
- Verifikationsrapport

First Office Activity

FOA är det slutliga testet som systemet skall genomgå. Här används ett komplett nätverk hos kund för att verifiera att alla system fungerar tillsammans. FOA utförs normalt som en separat aktivitet i projektet, om möjligt även med kund. Syftet är att visa att systemet lever upp till de krav och förväntningar som kunden har.

Aktivitet

- Ge FOA tillräckligt med support från utvecklingsprojektet
- Om möjligt, utför ett acceptanstest
- Rapportera problem
- Gå tillbaka till adekvata aktiviteter och genomför de förändringar som behövs.

Input

- Kravspecifikation
- Det verifierade systemet
- Produktinformation för kund
- Verifikationsstrategi

Consolidation

Slutförs vid Milstolpe 11 (MS11).

Aktivitet

- Lagra allting i GASK (General Archive System for Communication)

Input

- Dokumentationslista
- Produktstruktur

Conclusion

Slutförs vid Milstolpe 12 (MS12).

Aktivitet

- Avsluta och sammanställ process utvärderingen och se till att erfarenheterna från användandet av Darwin tas till vara.
- Lämna över produkten till TAC (Technical Assistance Center)

Output

- Alla värdefulla tillgångar som projektet använt skall föras tillbaka

3.4 Systemutvecklingsprocessen på Frontec

Frontec är ett konsultföretag som arbetar med alla slags system och de arbetar på alla typer av maskiner och operativ system.

Företag som Frontec samarbetar med är: Volvo (det första stora projektet efter att RUP togs i användning var den ’svarta lådan’ i Volvo S80 – man hade inte kunnat klara av ett så stort projekt med så mycket människor inblandade utan en modell som RUP och det verktygsstöd som följer med), Ericsson, SKF, Astra Hässle, kommun och landsting.
När Frontec slogs ihop med IDK ökade Frontecs kokurrensmöjligheter då Frontec å ena sidan har den tekniska kompetensen och IDK bidrog med den administrativa systemutvecklingen.

Frontec arbetar objektorienterat i den mån de själva får bestämma. Om kunden inte vill ha det så försöker Frontec ändå att i så stor utsträckning som möjligt arbeta objektorienterat i analys och design.

Frontec använder sig av RUP (Rational Unified Process) som generell systemutvecklingsmodell och de använder sig av PROPS[5] som projektstyrningsmodell.
RUP är nyinköpt och alla har inte hunnit sätta sig i det hela än, men det är en kvalitetspolicy att alla skall följa detta.

Anledningen till varför de valde RUP var att de ville ha en metod som fanns och var gångbar på marknaden och som kunderna använde sig av. Att använda sig av en sådan metod skulle ge Frontec en ’kvalitetsstämpel’.
Med dessa kriterier finns det idag inte många alternativ till RUP.

P.g.a att de bara använt sig av RUP under så kort tid, endast ett år, har de ännu inte kunnat se några disktinkta effekter. Det tar tid att arbeta upp kompetensen att använda RUP med förstånd, dvs. hur man kan anpassa metoden efter ens behov.

Pär Lagström på Frontec tror inte att användandet av RUP automatiskt kommer att innebära lägre utvecklingskostnader eller kortare leveranstid.
Han tror dock att det i framtiden kommer innebära högre kvalitet på systemen de bygger och detta ger nöjda kunder.

[5] PROPS är Ericssons egenutvecklade projektstyrningsmodell.

3.5 Systemutvecklingsprocessen på Ericsson Hewlett Packard Telecommunications

EHPT är ett företag som bygger programvara och utvecklar drifts- och affärsstödssystem för telenät.
Exempel på system som de utvecklar är:

OSS (Operations Support Systems) som är till för att övervaka och påverka trafiken i telenät främst vad som gäller telefoni både mobil och fast.
BSS (Business Support Systems) affärsstödssystem för att samla in samtalsdata för att kunna debitera, uppkopplingar mot banker osv.

Från att tidigare ha varit kunddrivna har de övergått till att vara marknadsdrivna dvs att de utvecklar produkter till marknaden istället för till en enskild kund.

EHPT använder sig av många speciella utvecklingsmetoder, några av dem är: PROPS, RUP och SDPM (System Development Process Model). SDPM håller dock på att försvinna till fördel för en heltäckande RUP tillsammans med PROPS.
De har också en metod som heter SLC (Software Life Cycle) som är den modell som Hewlett Packard använde sig av tidigare.
FAME är en egenutvecklad tollgatemodell som har drag av PROPS och SLC.

Projektets inriktning styr val av metod och metoden anpassas därefter.
Eftersom EHPT redan innan de valde RUP som systemutvecklingsmetod arbetade iterativt och objektorienterat såg de RUP som en passande metod för att formalisera arbetet.
PROPS och SDPM har de använt i ca 10 år, Objectory (RUP) har de använt sig av lika länge, men inte i lika stor utsträckning. Notationen inom EHPT´s systemutveckling är UML.

EHPT har inga problem med att arbeta objektorienterat för ’det bara finns där’ som en naturlig del av EHPT´s tankesätt. Strukturen på analys, design och programmeringsspråk är objektorienterad, men det finns dock fler sätt att se på strukturen (EHPT är så stort att det finns många grupper inom företaget som arbetar på olika sätt).

Det kan dock finnas problem med det iterativa arbetssättet då alla inte är överens om att detta är det bästa sättet att utföra systemutvecklingsarbetet på.

3.5.1 Rup

RUP (The Rational Unified Process) är en process för mjukvarukonstruktion som man kan använda över ’webben’ och som har en sökbar kunskapsbas. Målet med RUP är att kunna garantera hög kvalitet på producerad mjukvara som möter slutanvändarnas behov inom förutsagd tid och budget. RUP är utvecklat och underhålls av Rational Software. RUP skall öka produktiviteten i ett projektarbete då alla medlemmar i projektet har tillgång till samma kunskapsbas, vare sig man arbetar med krav, design, test, projektledning eller konfigurationsledning, och då använder sig av samma språk och syn på hur man utvecklar mjukvara.

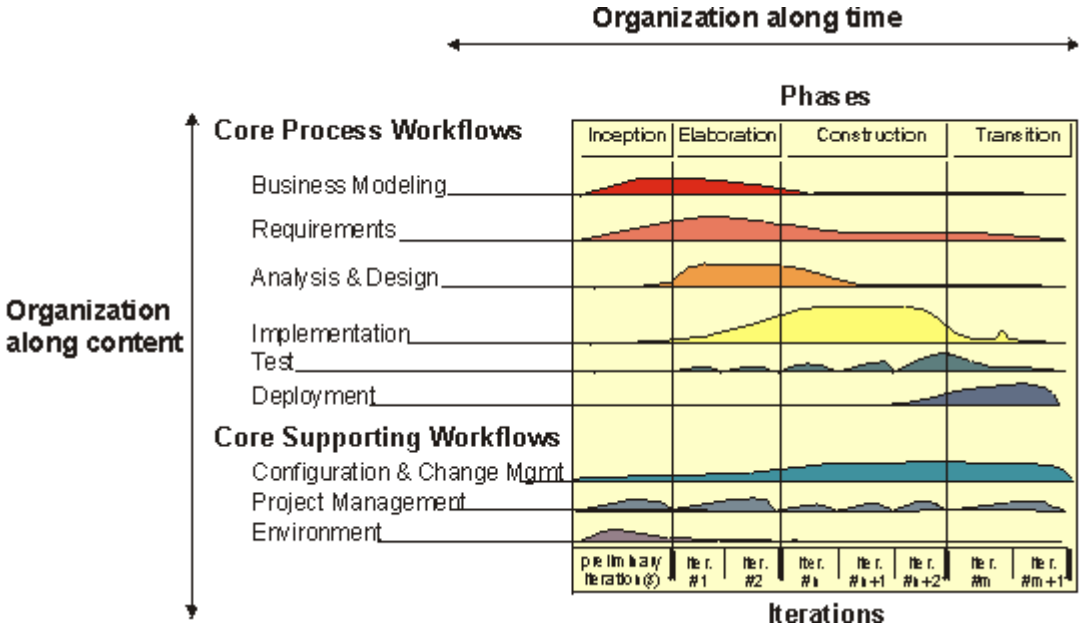
RUP är en guide för hur man effektivt skall kunna använda UML (Unified Modeling Language). Den stöds av verktyg som automatiserar stora delar av processen. Dom används för att skapa och underhålla artificiella modeller vad gäller visuell modellering, programmering, testning m.m.

Det är en anpassningsbar process som kan användas både till små och stora projekt. RUP beskriver hur man effektivt kan använda sig av kommersiellt bevisade tillvägagångssätt att utveckla mjukvara på. Dessa kallas för ”Best practises”. Exempel på dessa är:

1. Att utveckla mjukvara iterativt
2. Behandla risker
3. Använda komponentbaserade arkitekturer
4. Visuellt modellera mjukvara
5. Verifiera mjukvarans kvalitet
6. Kontrollera ändringar i mjukvaran

Processen kan beskrivas i två dimensioner, eller längs två axlar:

Den horisontella axeln representerar tid och visar den dynamiska aspekten av processen och uttrycks i cykler, faser, iterationer och milstolpar. Den vertikala axeln representerar den statiska aspekten av processen, hur den beskrivs i form av aktiviteter, arbetare och arbetsflöden.



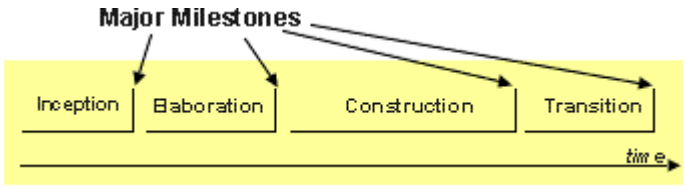
Figur 13

Faser och iterationer

Mjukvarans livscykel bryts ner i cykler. RUP delar in en utvecklingscykel i fyra faser:

Inception phase (påbörjande fas)
Elaboration phase (utvecklingsfas)
Construction phase (konstruktionsfas)
Transition phase (övergångsfas)

Varje fas avslutas med en väldefinierad milstolpe, där vissa kritiska beslut måste ha tagits och vissa huvudmål måste ha uppnåtts.



Figur 14

Inception phase

Under denna fas görs avgränsningar för projektet. För att åstadkomma detta måste man identifiera alla externa entiteter som systemet kommer att interagera med. Detta innefattar att identifiera alla ”use-cases” och beskriva de viktigaste av dessa.

Vid slutet av denna fas kommer den första stora milstolpen för projektet: the Lifecycle Objectives Milestone. Efter utvärderingen av denna kan projektet annulleras eller behövas omstruktureras om det inte passerar milstolpen.

Elaboration phase

Meningen med denna fas är att analysera källan till problemen, etablera en arkitektonisk grund, utveckla projektplanen och eliminera de största riskerna för projektet. Man skulle kunna säga att denna fas är den mest kritiska av de fyra faserna. I slutet av denna fas anses själva modelleringen av systemet färdig och man ska besluta om man ska börja på konstrueringsfasen eller inte. Det är nu det för många projekt kan uppstå höga risker och höga kostnader. Man ska från denna fas kunna förutspå ungefärliga kostnader och schema för slutförande av projektet.

I denna fas byggs en körbar arkitektprototyp i en eller flera av iterationerna, beroende på begränsningar, storlek och risk för projektet. Den stora milstolpen vid slutet av denna fas är Lifecycle Architecture. Projektet kan annulleras eller behövas omstruktureras om det inte passerar milstolpen.

Construction Phase

Under denna fas utvecklas alla kvarstående komponenter och applikationer som integreras i produkten och allt testas noggrant .

Transition phase

Meningen med denna fas är att installera produkten hos användaren. När produkten givits till slutanvändaren upptäcks ofta nya behov och saker som behöver ändras.

Denna fas fokuserar på de aktiviteter som krävs för att installera mjukvaran hos användaren. Denna fas innehåller ofta flera iterationer. Man lägger ner mycket möda på att få en användarorienterad dokumentation, utbilda användare och ge support till användarna.

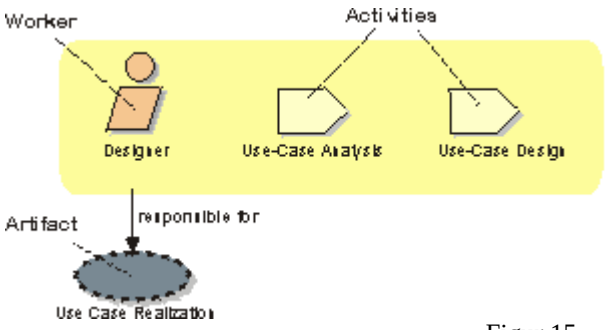
Milstolpen för denna är fas är release av produkten.

Iterationer

Varje fas i RUP kan i sin tur delas in i iterationer. En iteration är en komplett utvecklingsloop som resulterar i en release av en körbar produkt, en del av den slutliga produkten som är under utveckling, som växer inkrementellt från iteration till iteration för att till slut bli det färdiga systemet.

En process beskriver vem som gör vad, hur och när. RUP använder sig av fyra primära modelleringselement:

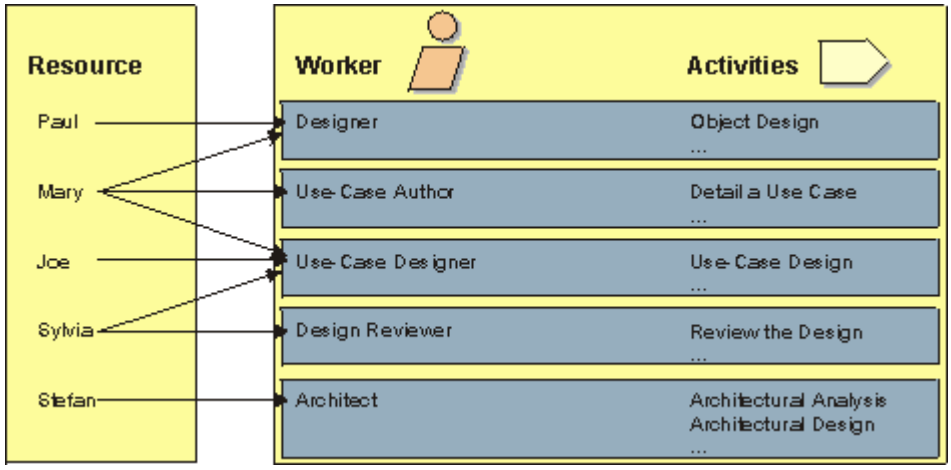
Workers – vem
Activities – hur
Artifacts – vad
Workflows – när



Figur 15

Worker

Worker definierar beteende och ansvar för en individ eller grupp av individer som arbetar tillsammans i ett team. En individ kan inneha många olika Worker-roller.



Figur 16

Activity

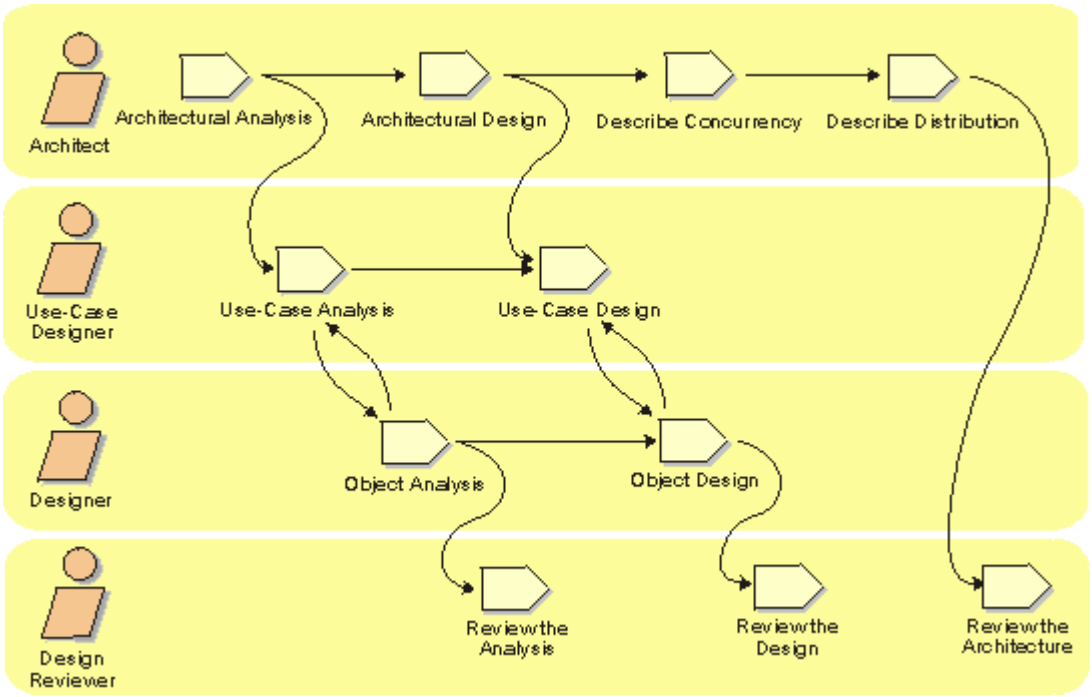
En specifik ”workers” aktivitet är en enhet av arbete som en individ i den rollen blir ombedd att utföra. Aktiviteten har ett klart syfte och uttrycks ofta i att skapa eller uppdatera en modell, klass eller plan. Varje aktivitet är tilldelad en specifik ”worker”. Exempel på aktiviteter är att planera en iteration, hitta ”use-cases” och aktörer, granskning av design och utföra tester.

Artifact

Artifact är en mängd information som är producerad, modifierad eller använd av en process. Artifacts används som input av workers för att utföra en aktivitet och är resultatet eller outputen av sådana aktiviteter. Exempel på artifacts är en modell som Use-Case modell, källkod och dokument som Business Case eller Software Architecture Document.

Workflows

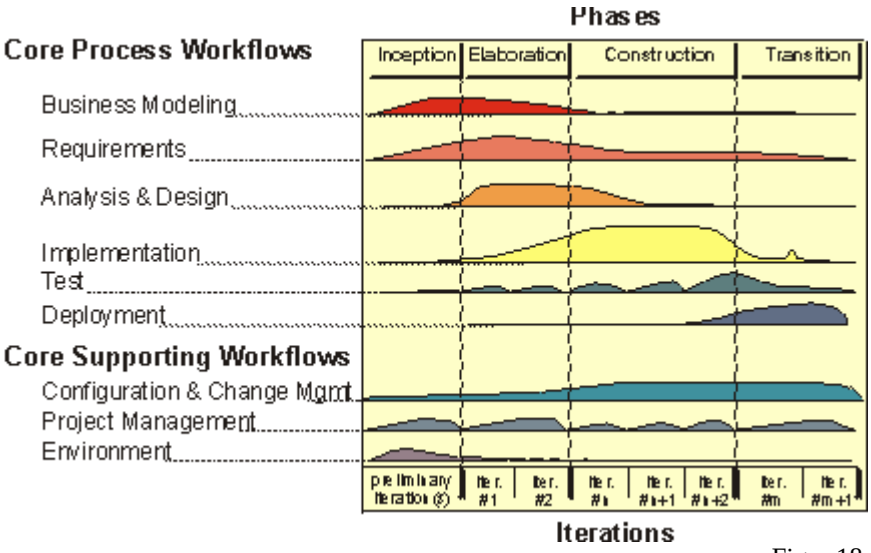
En workflow är en följd av aktiviteter som producerar ett resultat av märkbart värde. I UML-termer kan man uttrycka workflow med hjälp av sekvensdiagram, samarbetsdiagram eller ett aktivitetsdiagram.



Exempel på workflow , Figur 17.

Det bör noteras att det inte alltid är möjligt eller praktiskt att påvisa beroendena mellan aktiviteterna. Ofta är två aktiviteter mer sammanvävda än vad som visas, speciellt när de involverar samma worker eller individ.

Det finns nio huvudsakliga workflows i RUP som delar in och grupperar workers och aktivities.



Figur 18

Dessa delas i sin tur in i ”engineering workflows” och ”supporting workflows”.

Engineering workflows:

Business modeling workflow
Requirements workflow
Analysis and Design workflow
Implementation workflow
Test workflow
Deployment workflow

Supporting workflows:

Project Management workflow
Configuration and Change Management workflow
Environment workflow

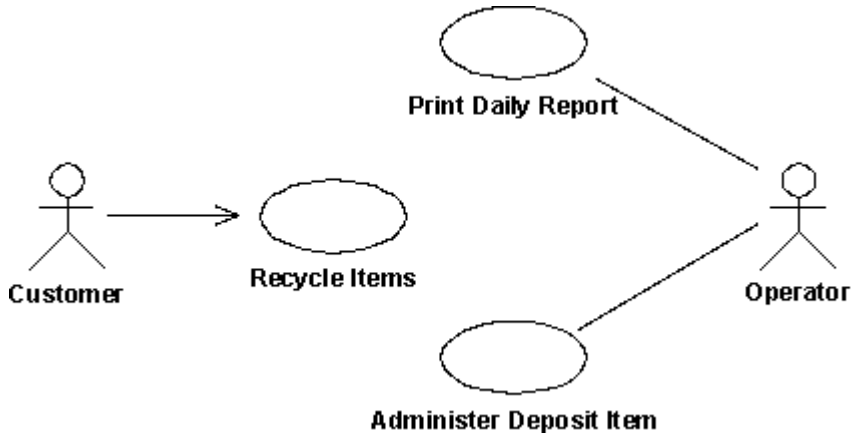
Även om namnen på de sex ”engineering workflows” påminner om de sekventiella faserna i en traditionell vattenfallsmodell bör man tänka på att faserna i en iterativ modell är annorlunda och att man går tillbaka till dessa gång på gång genom livscykeln.

Business Modeling (Verksamhetsmodellering)

Ett av de stora problemen med satsning på verksamhetsutveckling är att de som utvecklar mjukvaran och de som sysslar med verksamhetsutveckling inte kommunicerar så bra med varandra. Det leder till att det som kommer ut från verksamhetsutvecklingen inte används på rätt sätt som input i mjukvaruutvecklingen och vice-versa. För att underlätta detta använder man sig av samma språk och process för båda parterna. I Business Modeling använder man så kallade business use cases. Detta för att få gemensam förståelse för vilka verksamhetsprocesser som behöver stödjas i organisationen. Detta dokumenteras i en verksamhetsobjektmodell.

Requirements (Krav)

Målet med detta är att beskriva vad systemet skall göra och låta utvecklarna och kunden att komma överens på den punkten. För att åstadkomma detta söker och dokumenterar man nödvändig funktionalitet och restriktioner. Aktörer som representerar användarna identifieras och man identifierar även andra eventuella system som systemet under utveckling skall interagera med.



Figur 19

Analysis and Design (Analys och design)

Målet med detta är att visa hur systemet skall realiseras i implementationsfasen. Analys och Design resulterar i en designmodell och eventuellt en analysmodell. Designmodellen tjänar som abstraktion till källkoden, dvs den ligger till grund för hur källkoden skall struktureras och skrivas.

Implementation

Syftet med implementationen är att definiera strukturen på koden och lagren av subsystem.

Systemet realiseras genom implementation av komponenter. RUP beskriver hur man återanvänder redan existerande komponenter eller implementerar nya komponenter.

Test

Syftet med testning är att verifiera interaktionen mellan objekt, verifiera lämplig integration av alla komponenterna i mjukvaran och verifiera att alla krav har blivit implementerade korrekt. Eftersom RUP föreslår ett iterativt tillvägagångssätt görs tester genom hela utvecklingsprocessen. Detta innebär att man tidigare kan upptäckt eventuella defekter.

Deployment (Spridning)

Syftet med detta är att framgångsrikt producera produktreleaser och leverera mjukvaran till slutanvändaren. Det ingår ett flertal aktiviteter:

Producera externa releaser av mjukvaran
Paketera mjukvaran
Distribuera mjukvaran
Installera mjukvaran
Förse användarna med hjälp och assistans.

Även om dessa aktiviteter centreras kring övergångsfasen (transition phase) behöver man också inkludera aktiviteterna i tidigare faser för att förbereda spridningen i realiseringsfasen.

Project Management

Här balanseras de olika målen, man ska hantera risker och överkomma begränsningarna att leverera och komma fram till en produkt som tillfredställer både kunden (den som betalar) och användarna. Målet med denna del är att underlätta arbetet genom att sätta upp ett ramverk och riktlinjer för planering, deltagande och utförande av projektet. Det är även att sätta upp ett ramverk för riskerna.

Configuration and Change Management

I detta arbetsflöde beskrivs hur man ska kontrollera stora mängder av ’artifacts’ som producerats av ett flertal personer som arbetar inom ett gemensamt projekt. Hänsyn skall tas till att flera personer kan arbeta med en och samma ’artifact’ och det kan finnas ett flertal versioner av systemet.

RUP beskriver hur man skall hantera parallell utveckling och utveckling som gjorts på flera olika ställen. RUP beskriver också hur man ska hålla reda på om en ’artifact’ ändrat och i så fall varför och av vem.

Detta arbetsflöde täcker också inrapportering av fel som hittats och beskriver hur man skall hantera detta genom livscykeln.

Environment

Syftet med detta arbetsflöde är att förse organisationen som utvecklar mjukvara med en miljö för mjukvaruutveckling, både processer och verktyg, som behövs för att stötta utvecklingsteamet. Fokus läggs på aktiviteter för att konfigurera processen i form av ett projekt. Man fokuserar även på aktiviteter för att utveckla riktlinjer som stöd för projektet. Arbetsflödet för miljö innehåller även ett ’Development Kit’ som används för framtagning av riktlinjer, mallar och verktyg som är nödvändiga för att skräddarsy processen.

Integrering av verktyg

En mjukvaruutvecklingsprocess kräver verktyg för att stödja alla aktiviteter i ett systems livscykel, speciellt för att stödja utveckling, underhåll och bokföring av olika ’artifact’-modeller. En iterativ utvecklingsprocess ställer speciella krav på de verktyg du använder, så som bättre integrering mellan verktygen och koppling mellan modeller och kod. Man behöver också verktyg för att hålla reda på ändringar, automatisera dokumentation såväl som verktyg för att automatisera tester. RUP kan användas med ett flertal verktyg, både Rationals egna och andra.

4 Resultat

Alla metoder brukar innehålla faserna analys, design, realisering, implementering, test och underhåll, men på vilket sätt man väljer att bygga sitt system varierar.

Exempel på tillvägagångssätt vid systemutveckling är att använda sig av vattenfallsmodellen, en iterativ modell, inkrementell utveckling och/eller objektorientering.

För att få enhetlig notation under en systemutvecklingsprocess vill företagen ofta använda sig av en gemensam standard och ett exempel på sådan notation är UML.

Frontec och Ericsson Hewlett Packard Telecommunications använder sig av en inköpt modell, RUP (Rational Unified Process). Anledningen till att de valde RUP är att det idag inte finns så många alternativ som är gångbara på marknaden och som de flesta kunderna använder sig av. RUP har även tillhörande verktyg. Då vi inte har haft tillgång till all information kring RUP, kan vi inte svara på exakt hur detaljerad den är.

Binomen har en egenutvecklad modell, Luxus®. Då de kom i behov av en metod, dokumenterade de helt enkelt hur de gjorde och utvecklade därefter Luxus® vidare.

Luxus® är relativt detaljerad, då den beskriver exakt vilka dokument/produkter som skall produceras i alla aktiviteter. Den beskriver även tydligt vad det är som skall utföras och i vilken ordning, och med förslag till vilka verktyg som kan användas. Dock finns inga instruktioner om *hur* dessa resultat skall produceras, ex. Användningsfallsanalys. Här står det att man skall producera en användningsfallsbeskrivning, men Luxus® ger inga förslag till hur man bör gå tillväga för att hitta dessa användningsfall. Skall man använda sig av rollspel, prototyper eller ”vägg-grafer”?

Ericsson Mobile Data Design har utvecklat Darwin som de använder sig av. Även denna metod är en formalisering och dokumentering av hur det tidigare arbetet utfördes. Den har allt sedan dess utvecklats och omarbetats utifrån de erfarenheter man dragit av användandet. Darwin är mycket detaljerad och beskriver utförligt i vilken ordning man skall utföra arbetet. Den beskriver även vilka dokument som skall produceras. Darwin säger dock inget om objektorientering, utan lämnar detta öppet för projektets medarbetare att bestämma.

Cap Gemini har sin egenutvecklade PERFORM, där vi har tittat på modulerna för utveckling av affärssystem och iterativ applikationsutveckling. PERFORM bygger på ”Best practise”, dvs dokumentering och formalisering av bra arbetsmetoder.

PERFORM är mycket detaljerad, den beskriver steg för steg exakt hur man skall gå tillväga i utvecklingsprocessen. Den beskriver även ingående vilka dokument som skall produceras i varje fas och vad som skall föras vidare till nästa fas.

4.1 Jämförelse

För att kunna dra egna slutsatser har vi gjort jämförelser mellan iterativt arbete och vattenfall och vi har också jämfört de företag som vi besökt. Vi har delat in företagen i

kundnära och marknadsnära. Andra aspekter som vi tagit med är ålder, storlek och huruvida de arbetar objektorienterat eller inte.

4.1.1 Iterativt kontra vattenfall

Fördelarna med att arbeta enligt vattenfallsmodellen dvs sekventiellt är att man hela tiden vet exakt var i utvecklingsprocessen man befinner sig. Det är lättare att få en översikt över projektets budget då man vet vid vilken tidpunkt som vilka resurser behövs. Iterativt tillvägagångssätt däremot har fördelarna att man tidigare upptäcker risker och det är lättare att göra förändringar, man har större möjlighet att återanvända kod och projektteamet utvecklas och lär sig under arbetets gång. Man har även prototyper under processen som användarna kan få utvärdera och man kan då tidigare hitta brister i systemet som man omedelbart kan åtgärda.

4.1.2 Jämförelser mellan företagen

Företag	Metod	Ålder	Storlek	Iterativt/sekventiellt	Objektorient	Notation
Cap Gemini	Perform		20 länder/ 38000 anst.	Båda	Kan anpassas	
Frontec	RUP m.fl.	18 år	10 länder/>1000 anst	Iterativt	Ja	UML
Binomen	Luxus®	6 år	Göteborg/ 20 anst	Sekventiellt	Ja	UML
EHPT	RUP m.fl.	6 år	8 länder/ 1000 anst	Iterativt	Ja	UML
ERV	Darwin	~19 år	Göteborg/ 400 anst	Sekventiellt	Kan anpassas	UML

Vi har ställt upp de kriterier vi har jämfört efter i tabellform för att på ett mer överskådligt sätt presentera underlaget för våra slutsatser.

4.2 Diskussion/Slutsatser

De slutsatser vi kan dra av vår undersökning är att företagen inte använder sig av en renodlat iterativ eller sekventiell metod. Darwin är till sin struktur sekventiell, men aktiviteterna i sig är iterationer. PERFORM består av flera moduler varav en modul, IAD, beskriver ett iterativt arbetssätt som man kan tillämpa på olika projekt och en annan modul, Business Application Reference Manual, är huvudsakligen sekventiell.
Frontec och EHPT använder sig av RUP som är iterativ. Sex av de nio stora arbetsflödena i RUP påminner om de sekventiella faserna i en traditionell vattenfallsmodell, men man bör man tänka på att faserna i en iterativ modell inte är exakt de samma som i en vattenfallsmodell.

Binomens Luxus® är sekventiell men där momenten kan genomföras iterativt. Luxus® har fått en sekventiell struktur för att underlätta projektstyrningen, men Binomen använder sig samtidigt av sk. “återkopplande pilot-användningsfall” där man väljer ut ett eller ett par representativa användningsfall som man kör i botten, inkluderande implementation och test, innan man slutför det resterande analys och designarbetet.

Kan man då dra några paralleller mellan metoderna och företagets storlek? Utifrån det material vi har haft tillgång till kan vi se att Cap Gemini, Ericsson Mobile Data Design och Ericsson Hewlett Packard Telecommunications har metoder som är mer komplexa (i jmfr med Luxus®) och som lämnar öppningar för anpassning till så gott som vilka

projekt som helst. Metoderna i större företag är viktiga för att ge en bra kommunikation över alla avdelningar. Detta är inte riktigt fullt så viktigt på ett mindre företag där man har bättre möjligheter till direktkontakt och där man har en större överblick över alla företagets projekt. I de större företagen är det även viktigare att man har mer detaljerade beskrivningar för hur man skall gå tillväga i en utvecklingsprocess så att man får ett enhetligt arbetssätt. Det kan annars uppstå problem vid integrationen av de olika delarna i projekten och skapa kommunikationssvårigheter.

Vad gäller åldern på företagen har vi inte kunnat se några direkta paralleller, men dock har vi under våra intervjuer fått den uppfattningen att t.ex. Cap Gemini, som är det äldsta av de företag vi besökt, inte är lika öppna för att helt övergå till objektorientering och iterativt tillvägagångssätt. Detta har ju också att göra med storleken på företaget att göra, det är ju inte fullt så lätt att styra in 38000 anställda på ett nytt tankesätt, som det är på t.ex Binomen med bara 20 anställda.

Om man ser på företagen som kundnära/marknadsnära, har vi inte heller här hittat några distinkta paralleller. Vi hade från början inställningen att de företag som arbetar kundnära borde ha större nytta av ett mera iterativt arbetsätt då detta gör det enklare att arbeta utifrån föränderliga krav. Man får ju då även fram prototyper som leder till att beställaren/kunden har en större möjlighet att kommunicera kring produkten och hitta felaktigheter eller nya behov. Man kan dock säga att t.ex. Frontec och Binomen som arbetar kundnära även arbetar iterativt , även om de inte uteslutande arbetar på detta sätt.

4.3 Egna kommentarer

Vi inser att arbetet kunde ha gjorts mer utförligt genom att ändra våra metoder.
T ex genom att välja företag mer noggrant, kategorisera dem bättre, prova metoderna på fiktiva projekt, mera ingående analysera/studera effekterna av användandet av metoderna på företagen.

Hade vi tittat på dessa aspekter så hade vi kanske kunnat komma fram till mer specifika riktlinjer av vilka metoder som bör användas vid vilket tillfälle, vilka effekter metoderna skulle kunna ge samt ge en mer rättvis bedömning av de analyserade metoderna.

5 Referenslista

1. Brown, David (1997). An introduction to Object-oriented analysis: objects in plain English, Wiley cop.

2. Avison, D.E., Fitzgerald, G. (1995). Informations systems development: Methodologies, techniques and tools, McGraw-Hill.

3. Sommerville, Ian (1996). Software Engineering, Addison-Wesley cop.

4. Apelkrans, Mats, Åbom Carita (1988). Systemering, Lund Studentlitteratur.

6 Bilagor

1. Frågeformulär

2. Darwin, Design dokument

3. Darwin, Prestudy design information table

4. Darwin, Feasibility design table

5. Darwin, System and subsystem design information table

6. Darwin, Lower level design infornation table

Bilaga 1

Frågeformulär till NN angående systemutvecklingsmetoder/modeller på NN.

1. Använder sig företaget av någon speciell utvecklingsmodell och i så fall vilken?

2. Är modellen en standardmodell eller är den företagsutvecklad?

3. Om modellen är företagsutvecklad, vilken/vilka teoretiska modeller bygger den på?

4. Hur används modellen i systemutvecklingen? Följer man modellen helt och hållet eller har man den bara som rättesnöre?

5. Vad för slags system är det företaget utvecklar?

6. Vilka kriterier hade ni när ni valde modell?

7. Hur länge har ni använt modellen?

8. Har er systemutvecklingsmodell effektiviserat arbetet på ert i företag och i så fall hur?

Bilaga 2

Technical Report

Contents

The Technical Report is a description of an investigation into a specific area. It may be a separate document, or an appendix to an IS or IP.

Properties

- Type: Project document
- Decimal class: 0360
- Approved by: Project Manager
- Approved at: MS1, MS2
- Written when appropriate

Function Description, FD

Contents

The FD defines how the functionality of a system function is distributed over the product implementation structure, i.e. what is implemented where. It specifies the transition from a function approach to an design unit approach. The FD is the last design document with a function approach.

There is a 1:1 relation to RS.

Properties

- Type: Product document
- Decimal class: 155 16
- Approved by: Design Responsible
- Approved at: MS3
- Written once per function or function group

Feature Description

Contents

The Feature Description contains the same type of information as an RS, but compiled for customer use.

Feature Descriptions are only written when there are external needs for this information. The document type do not serve any purpose for the project's internal work.

Properties

- Type: Product document
- Decimal class: 1550
- Approved by: Design Responsible
- Approved at: MS3
- Written once per project, function or function group

Design Specification, DS

Contents

The DS specifies the implementation of a design unit. If the design unit is further decomposable, the DS contains sequence diagrams that describes the interactions between lower level units. These sequence diagrams describes the implementation of use cases.

The DS constitutes the complete description of the design unit implementation and thus contains sequence diagrams and design architecture descriptions for every function and use case the design unit implements parts of.

Properties

- Type: Product document
- Decimal class: 102 62
- Approved by: Design Responsible
- Approved at: MS4, MS5
- Written once per design unit

Interface Description, IFD

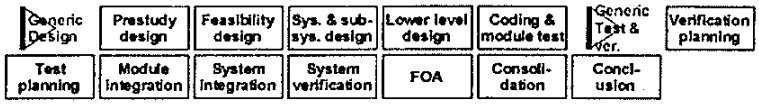
Contents

The IFD specifies the detailed interface of a design unit, i.e. signal definitions and method parameters. If the design unit is a common service or library there should be extensive descriptions on usage. If

Properties

- Type: Product document
- Decimal class: 155 19
- Approved by: Design Responsible
- Approved at: MS4, MS5

Design Documents



This page presents all documents needed in conjunction with design.

The documents are:

- Requirement Specification
- Implementation Sketch
- Technical Report
- Function Description
- Feature Description
- Design Specification
- Interface Description

Design in Darwin:

Generic design activity	Describes the workflow in generic terms
Design documents	List of all design documents
Structures	Describes the connection between design and verification

Requirement Specification, RS

Contents

The RS specifies system level requirements in terms of general requirements (characteristics, implementation etc...) and functional requirements broken down to the level of use case descriptions.

Each requirement or group of related requirements is explained with a motivation text.

The RS is maintained during the project, preferably with tool support (Rational Requisite Pro).

It is recommended to cover the product's complete set of requirements.

Properties

- Type: Project/Product document (depending on how they will be handled in relation to the product lifecycle)
- Decimal class: 102 91
- Approved by: Product Management
- Approved at: MS1
- Written once per project or once per function area (or similar)

Implementation Sketch, IS

Contents

The IS is a Prestudy document, used for documenting design ideas and estimations on the implementation work.

Properties

- Type: Project document
- Decimal class: 1594
- Approved by: Project Responsible
- Approved at: MS1
- Written per function or function group

Implementation Proposal, IP

Contents

The IP is a Feasibility Study document, used for documenting design proposals and estimations on the implementation work for those proposals. It is a refinement of the information in ISes.

Properties

- Type: Project document
- Decimal class: 159 41
- Approved by: Project Responsible
- Approved at: MS2
- Written per function or function group

Bilaga 3

Prestudy design information table

Feasibility design information table ▷

Actions to perform	Information headlines / categories	Documented in...
Design aspect: Function (Prestudy design)		
Define functions and subfunctions. Identify product improvements and known faults.	Connection between functional requirements and functions and subfunctions. ReqPro will contain complete information about the requirements, including status. The RS will also contain motivations for each requirement.	ReqPro & RS
	Use case outline (These are verbal descriptions of the most important use cases, not sequence diagrams. It is important to keep the descriptions on an overall level. See the design flow picture in the Structures chapter.)	ReqPro & RS
Specify the functions and their verbal use cases on the system level. Trace the requirements by pointing from each use case to the affected requirements. Point into the frozen requirements documents (RS + CR). Do not create a Rose model on the system level.	Verbal use case descriptions	ReqPro & RS
	Actor descriptions	RS
	Characteristics of the functions (performance etc.)	RS
	Characteristics of the products (power and voltage, FCC, etc.)	RS
Design aspect: Distribution (Prestudy design) ▷		
Make a rough definition of the implementation architecture if this does not already exist. Distribute the functions across this rough model. (This is based on the most important use cases. As the analysis proceeds in the next activities, the number of use cases included in the analysis increases.)	Influence on the implementation or a preliminary implementation architecture	Implementation Sketch
	Time and cost estimations (This is done for each requirement, not for a certain implementation solution. It is very preliminary, possibly with factors of uncertainty. The estimates shall cover both the Feasibility Phase and the whole project.)	Implementation Sketch
Design aspect: Implementation design (Prestudy design) ▷		
Ensure that critical parts are possible to implement. This is done on a *rough* level of detail.	Critical aspects of the implementation	Technical Report (may be an appendix to the IS)
Design aspect: Prototyping/simulations (Prestudy design) ▷		
Investigate critical parts (to a limited extent).	Prototyping of critical aspects of the implementation	Technical Report (may be an appendix to the IS)
	Verification that requirements on	Technical Report

	critical implementation parts are reasonable (through simulations)	(may be an appendix to the IS)
Design aspect: Coding (Prestudy design)		
Identify any requirements that will affect coding. These may be coding requirements, standards, portability, etc.	Implementation requirements	ReqPro & RS
Design aspect: Test and verification (Prestudy design)		
Ensure that the products will be possible to verify. This is done on a *rough* level of detail.	Descriptions on how each requirement will be verified.	ReqPro & RS
Design aspect: Productification (Prestudy design)		
When refining an existing product: Define the design base. Identify which products that will be affected by the project. Identify which new products that may be created. This is done on a *rough* level of detail.	Design Base	Implementation Sketch
Document the connections between the defined functions graphically.	Preliminary functional product structure (including functions and subfunctions)	Product Structure (1315)
Identify potential third party products.	Sourcing policy. Third party products	When there are requirements on this: ReqPro & RS
Design aspect: Industrialisation (Prestudy design)		
Roughly identify which new products that may be sold, and how the deployment will be done. (For example: will there be software keys to the products? Do we plan to sell 1 or 100 units? Define the production volume. This information is to be used when planning the industrialisation flow.)	Delivery volumes	ReqPro & RS

Feasibility design information table

◁ Prestudy design information table System and subsystem design information table ▷

Actions to perform	Information headlines / categories	Documented in...
◁ Design aspect: Function (Feasibility design) ▷		
Identitify the functional documentation to be produced. Also further identify product improvements and known faults. During this activity, the information to be handled is the same as in Prestudy design. However, it is taken to a deeper level of detail and refinement.	See Prestudy design.	See Prestudy design.
◁ Design aspect: Distribution (Feasibility design) ▷		
Define interfaces, protocol layers (only *which* they are), and have an idea about application protocols. Analyse how the implementation parts will be affected. This is done by expanding the number of use cases with those seen as having the "second most important" influence on the implementation architecture.	(Do the same things as in System & Subsystem Design, but with a more general scope. A deeper analysis is only made for critical parts. Each project decides when a satisfying depth and coverage is reached.)	Implementation Proposal
	Time and cost estimations	Implementation Proposal
Make a *rough* anatomy plan (see System and Subsystem Design).	Anatomy Plan	Project Specification
◁ Design aspect: Implementation design (Feasibility design) ▷		
Do the same things as in System and Subsystem Design, but only for critical parts. As a basis for cost estimations, there must also be an overall analysis of the whole system.	(See System and Subsystem Design)	Technical Report
Define which design documents to be created (update existing lists if there are any).	Design documents	Document List
◁ Design aspect: Prototyping/simulations (Feasibility design) ▷		
Do the same things as in Prestudy Design, but deeper and with a wider scope.	See Prestudy Design.	See Prestudy Design.
◁ Design aspect: Coding (Feasibility design) ▷		
Identify design rules. This means to find out in which areas we have them, and in which area we don't. It does not include to create what is lacking.	Design rules	Technical Report
Identify languages and development environment. Also identify training needs in this area.	Languages and development environment	Technical Report
◁ Design aspect: Test and verification (Feasibility design) ▷		
Define and make cost estimations for new test tools. See also Verification	Cost estimations of test tools.	Implementation Proposal for test tools

planning.		
◁ Design aspect: Productification (Feasibility design) ▷		
Begin creating the structures to be used. The functional product structure should be ready. (It is not yet defined to what level this is done, if it means that product numbers are defined for all elements or not.)	Functional product structure	Product Structure (1315), Structure Specification (131 61)
	Preliminary design structure (the elements do not yet have to be numbered in this structure).	Product Structure (1315), Structure Specification (131 61)
Identify the lowest level of deliverables (for example if the node software will consist of one package or several parts). Make a delivery plan for documents and products.	Delivery structure (it is not yet defined how detailed it should be at this point).	Product Structure (1315), Structure Specification (131 61)
When designing new products: Identify which products in the structure that may be reused parts or third party products. This does *not* include selecting suppliers.	Sourced products, Licenses, Reuse	Implementation Proposal
◁ Design aspect: Industrialisation (Feasibility design) ▷		
Identify the scope of our responsibilities from a delivery and packaging point of view. Identify build routines and specify tools for builds and production.	Production and manufacturing activities	Project Specification

System and subsystem design information table

◁ Feasibility design information table Lower level design information table ▷

Actions to perform	Information headlines / categories	Documented in...
◁ Design aspect: Function (System and subsystem design) ▷		
During this activity, the information to be handled is the same as in Prestudy design. However, it is taken to a deeper level of detail and refinement. If necessary, gather information for customers about the functions to be implemented.	Customer oriented function descriptions.	Feature Description
	Customer oriented subfunction descriptions.	Feature Description
◁ Design aspect: Distribution (System and subsystem design) ▷		
For each function: Start with the preliminary design structure. Select design elements and distribute functions and subfunctions to them. Do this work in several iterations Describe the distribution per function. Connect system functions (collections of use cases) to the subsystem level in the design structure. This connection is visualized in the design flow picture in the Structures chapter. Develop the final design structure on system and subsystem level. This includes identifying interfaces to be used. This is the point where requirements tracking stops during design. It comes back at verification.	Package & Class descriptions (This is done on the level *below* the one you are currently on. On system level, you describe the subsystem packages, on subsystem level you describe the classes below that.)	Function Description, Rose
Ensure designer commitment by creating an anatomy plan. This is a detailed delivery plan showing the connection between project increments and the product. In other words, it shows the work of the design teams. The anatomy is primarily function oriented. It contains services that can be verified at System Integration / Verification. It is also possible to include design elements that must be ready at an early time. These can however not be verified at SI/SV.	Anatomy Plan	Anatomy Plan
Start with use cases at system level. Create sequence diagrams showing node interaction for each use case.	Sequence diagrams	Rose, Function Description
	Collaboration diagrams	Rose, Function Description
Create sequence diagrams showing internal interactions at the top level within each node for each use case. Describe this in one document per node.	Sequence diagrams, Collaboration diagrams	Design Specification, Rose
◁ Design aspect: Implementation design (System and subsystem design) ▷		

Briefly describe the methods in the classes.	Method descriptions (not always necessary)	Rose or Design Specification & Interface Description (on subsystem level)
◁ Design aspect: Prototyping/simulations (System and subsystem design) ▷		
When necessary, do the same as in Prestudy Design.		
◁ Design aspect: Coding (System and subsystem design) ▷		
When updating existing products: Read existing code in order to understand the best way to add new functions.	Design rules and coding conventions	Design Rules
◁ Design aspect: Test and verification (System and subsystem design) ▷		
Identify how testability will be ensured, and how this affects the design rules.	Testability description and coding conventions	Design Rules
◁ Design aspect: Productification (System and subsystem design) ▷		
Define how each product is to be handled (examples: using ClearCase, Delta, the web, GASK, trouble reporting, release rules, status rules, etc.) Define how configurations will be made. This includes which possible combinations to be allowed and how to handle this. (This means e.g. a software key for each function versus separate modules, minimum and maximum configurations).	Productification description, including the following: • release handling • use of status codes • trouble reporting routines • archiving (including archives during the work) • product structure • configurability	T.B.D.
Finalise the design structure.	Design structure (the elements are numbered).	Product Structure (1315), Structure Specification (131 61)
◁ Design aspect: Industrialisation (System and subsystem design) ▷		
Define the logistics. This includes how the product is packaged, from our development environment all the way to our end customer.	(This is defined as a function, possibly with subfunctions and use cases for this.) • delivery routines • packaging routines • configurations to be handled	T.B.D.

Bilaga 6

Lower level design information table

◁ System and subsystem design information table

Actions to perform	**Information headlines / categories**	**Documented In...**
◁ Design aspect: Function (Lower level design)		
If necessary, include additional functions to the design elements (functions not coming from the functional structure). This means that they are black box descriptions of the design elements (example: several applications co-existing in the same node).	Functional specifications for the top level of the detailed design. (This means "application" in Network Management, "subsystem" in network elements. Make one description per function, which is a group of use cases.)	ReqPro & Function Specification (for subsystems)
◁ Design aspect: Distribution (Lower level design)		
Describe how the functions (and use cases) affect the design elements. Break down the description to the smallest manageable design element. Define interfaces (including the specification of APIs, function calls, flows and PDUs). At this point it is possible to begin to go lower than the lowest level in the design structure (i.e. to point at individual source code files).	Classes/modules (define which they are) - inheritance - relations	Rose (Design Specification)
	Sequence diagrams per use case (This is done on class/module level. No support for hierarchy in the sequence diagrams.) Collaboration diagrams (on the corresponding level) (These are optional. Use them when they add value, otherwise don't.)	Rose (Design Specification)
	State diagrams	Rose (Design Specification)
	Interfaces on "block" level (This can be sequence diagrams or verbally in broad outlines. In Unix / Network Mgmt this can for instance be process interfaces.)	Design Specification
◁ Design aspect: Implementation design (Lower level design)		
Describe the detailed design in broad terms. The elements on the lowest level are described in such a way that designers of average experience easily can understand them. In order to keep this effort at a reasonable level, make sure these descriptions are not as detailed as the code itself will be. Also describe how the design elements will be built together to form the nearest level above (e.g. within a subsystem).	Dynamic module descriptions (State machines - this may be done in SDL, UML, or verbally)	Design Specification
	Database modelling	e.g. Oracle Designer
	Method descriptions (This is e.g. function headers)	Source code files, Interface files, (Rose)

◁ Design aspect: Prototyping/simulations (Lower level design)		
Critical parts or alternative solutions may be prototyped. Ideas may be tried, but at this point it should be possible to use them as the basis for the actual product.	-	-
◁ Design aspect: Coding (Lower level design)		
When updating existing products: Continue to read existing code in order to understand the best way to add new functions. Create skeletons for the code, including function declarations for the important functions.	Code skeletons	Source code files and header files
Identify how CM will be affected (e.g. ClearCase and build routines). Define a trouble reporting project in DDTs (finished at MS6 at the latest).	Config spec	ClearCase
◁ Design aspect: Test and verification (Lower level design)		
When designing each module, take the Test and Verification Strategy and Test Plan into consideration. Consider how the tests are to be performed. Should test functionality be designed into the module or not? (Print debug info, log files, etc.) Will this remain in the product after delivery? When updating existing products: Is there existing support for test functionality? Will this be updated?	Test functionality	Design Specifications are affected.
◁ Design aspect: Productification (Lower level design)		
Status handling, when applicable.	Status information	(Where status information is defined)
Develop configuration rules.	Configuration rules	Structure Specification (13161)
◁ Design aspect: Industrialisation (Lower level design)		
Develop instructions and identify tools to be used when "producing" the products.	Code generation, Build instructions	Code Generation Information
Create/update information about the products that can be ordered (which they are and how they can be ordered).	Ordering Data, Ordering Information	Ordering Data, Ordering Information