

# 1 Problemformulering

Hvilke kvalitetsbegreber findes i XP og hvordan er disse begreber anderledes end kvalitetsbegreber fra de mere traditionelle systemudviklingsmetoder?

Først vil jeg give en kort, generel introduktion af XP og derefter vil jeg sammenligne XP med de traditionelle systemudviklingsmetoder<sup>i</sup>.

Jeg vil fokusere på kvalitetsbegreber i XP og de traditionelle systemudviklingsmetoder. Desuden vil jeg at komme ind på hvilke overvejelser, der ligger bag ved disse begreber. Jeg vil ikke beskæftige mig med måling af kvalitet vha. disse begreber, hvilket skyldes, at de som regel ikke kan måles objektivt. Man vil heller ikke kunne angive et kvalitetsniveau, som skal overholdes. Eller som Dahlbom skriver i CiC<sup>ii</sup>: "Quality is no attribute; it is a challenge to transcend".

Essayet er skrevet ud fra 'Introduktion til Extreme programming' [Kent2000] og teorien er hentet fra 'Computers in Context' (CiC) [Dahlbom1993]. Jeg har ikke selv praktisk erfaring med XP, men har fulgt med i debatten omkring XP, som har kørt de senere år, bl.a. i Computerworld.

---

Morten Nobel-Jørgensen  
17. november 2002, København

## 2 Grundprincipper i XP

### **Risiko: Det grundlæggende problem**

Styring af risiko er det helt store problem i softwareudvikling i dag. Der findes et utal af eksempler, fra medierne, om strandede og mislykkede projekter, hvor deadlines er overskredet, systemer er fejlbehæftet og prisen er højere end den først aftalte (ofte pga. ændrede krav til systemet). Ud over de kendte historier, som oftest er fra det offentlige, må der være et endnu større antal mislykkedes projekter fra det private erhvervsliv, som aldrig har nået offentlighedens lys.

Følgende viser en kort oversigt over de største risici softwareudvikling og hvordan XP angriber disse:

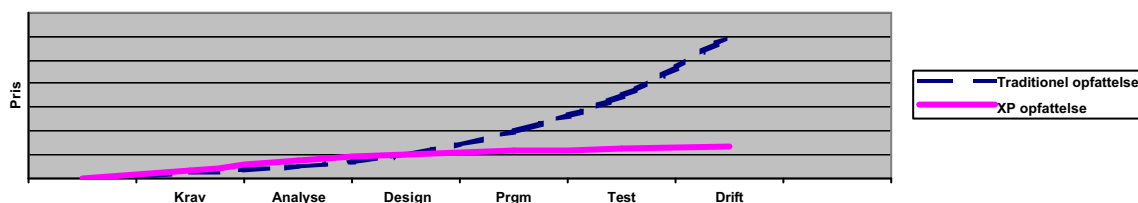
Risiko	XP's håndtering af risiko
Overskredne deadlines	<ul style="list-style-type: none"> <li>◇ Hyppige afleveringer med højst et par måneder mellem hver.</li> <li>◇ Først aflevering indeholder den grundlæggende funktionalitet og kan sættes i drift. Evt. ændringsforslag bliver rettet i senere afleveringer.</li> <li>◇ Den højest prioriterede funktionalitet bliver færdiggjort først. Hvis tidsplanen ikke holder, er det dermed den mindst vigtige funktionalitet man må skippe.</li> </ul>
Projektet bliver lukket	<ul style="list-style-type: none"> <li>◇ Kunden skal definere den mindst mulige funktionalitet (med størst nytte). Dermed er der ikke så meget der kan gå galt.</li> </ul>
Systemet sander til	<ul style="list-style-type: none"> <li>◇ Skaber og vedligeholder en omfattende samling af testprogrammer.</li> <li>◇ Hver programændring, vil blive testet op imod disse testprogrammer, hvilket sikre at programmet altid er på et velfungerende stadie.</li> </ul>
For fejlbehæftet	<ul style="list-style-type: none"> <li>◇ Test af systemet fra både udviklernes og kundes synspunkt.</li> <li>◇ Unit-test, skrevet af udviklerne, sikre høj kvalitet af koden.</li> <li>◇ Funktionalitets-test, skrevet/defineret af kunden, sikre hele systemets funktionalitet.</li> </ul>
Misforståede behov	<ul style="list-style-type: none"> <li>◇ Kunden indgår i udviklingsgruppen. Specifikationen vil løbende blive udviklet gennem forløbet og afspejle de ting, som kunden og udviklerne lærer under vejs.</li> </ul>
Ændrede behov	<ul style="list-style-type: none"> <li>◇ Ved hyppige delafleveringer og kunden siddende med i projektgruppen vil ændrede behov blive afdækket løbende.</li> </ul>

Tabel 1 XP's håndtering af risiko

### **Ændringers pris**

Hele grundtanken i XP hviler på følgende antagelse; ændringers pris ikke behøver at stige voldsomt over tid.

Dette bryder med den traditionelle opfattelse; nemlig at ændringens pris er eksponentielt stigende over tid.



Figur 1 Ændringers pris

Ændringers pris kan reduceres ved:

- Benytte bedre programmeringssprog, bedre databaseteknologi og bedre værktøjer, som understøtter løbende refaktoring<sup>iii</sup>.
- Bedre programmeringsvaner. Enkelt og gennemskuelig kode er lettere at rette i. Test-værktøjer giver desuden et hurtigt overblik over de konsekvenser en ændring i koden måtte have.

For at XP kan fungere, kræves det, at ændringers pris holdes lav, da XP er en evolutionær og iterativ måde at udvikle systemer på, hvor ændrede krav til systemet er en naturlig del af projektet.

Dermed ikke sagt at ændringers pris altid kan holdes lav. Der kan være en række faktorer (fx gammel teknologi), som gør at ændringens pris er eksponentielt stigende over tid, og i disse tilfælde kan XP ikke bruges.

### **Dogme reglerne**

XP er på ét punkt væsentligt anderledes end andre systemudviklingsmetoder; XP definerer ikke et sæt af aktiviteter, som tilsammen udgøre en systemudviklingsmetode. I stedet er XP et sæt af enkle regler, som man ved at efterleve løfter kvaliteten af softwareudviklingen.

#### **Regler i XP fra [Kent2000]**

- ◇ Planning game: Vedtag hurtigt omfanget af den næste aflevering ved at kombinerer forretningsprioriteringer med tekniske estimater. Når virkeligheden overmander planen, så opdater planen.
- ◇ Hyppige afleveringer: Sæt hurtigt et enkelt system i drift, og frigiv derefter nye versioner med korte mellemrum.
- ◇ Metafor: Giv al udvikling en fælles retning med en fælles historie om, hvordan hele systemet virker.
- ◇ Enkelt design: Systemet skal på ethvert tidspunkt have et så enkelt et design som muligt. Overflødig kompleksitet skal fjernes, så snart den opdages.
- ◇ Test: Udviklerne skriver hele tiden unit-test, der skal køre fejlfrit, for at udviklingen kan fortsætte. Kunden skriver test, der viser, at funktionaliteten er færdigudviklet.
- ◇ Refaktoring: Udviklerne omstrukturerer systemet uden at ændre på dets opfattelse ved at fjerne gentaget kode, forbedre den pædagogiske fremstilling, forenkle eller tilføje fleksibilitet.
- ◇ Parprogrammering: Al produktionskode bliver skrevet af to folk ved hver maskine.
- ◇ Fælles ejerskab: Enhver udvikler kan, når som helst, rette kode, hvor som helst i systemet.
- ◇ Løbende systemintegration: Integrer og byg systemet mange gange hver dag, nemlig hver gang en opgave er færdiggjort.
- ◇ 40 timers arbejdsuge: Arbejd ikke mere end 40 timer om ugen som en grundregel. Arbejd ikke over to uger i træk.
- ◇ Kundetilstedeværelse: Tag en rigtig levende bruger med i gruppen til at stå til rådighed for spørgsmål på fuld tid.
- ◇ Kodestandarder: Udviklerne skriver al kode i overensstemmelse med et sæt regler, der lægger vægt på kodens formidlende rolle.

### **3 Kvalitetsbegreber i XP**

Jeg vil i dette afsnit se XP fra 2 forskellige perspektiver; Først vil jeg sammenholde XP med de tre store systemudviklingsparadigmer (konstruktion, evolution og intervention), for at se på de væsentligste forskelle.

Derefter vil jeg se på systemudviklingsmetoder som kvalitetskontrol. Ved at sætte dette perspektiv vil jeg finde, hvilke fokusområder på kvalitet XP har, sammenlignet med de traditionelle systemudviklingsmetoder.

XP ligner ikke på alle punkter en systemudviklingsmetode: XP er mere regelbaseret end aktivitetsbaseret. Alligevel vælger jeg at betragte XP som en systemudviklingsmetode, da den angiver, hvordan man systemudvikler<sup>iv</sup>.

## ***Fra konstruktion til evolution***

XP er en åbenlys modreaktion på konstruktionsparadigmet.

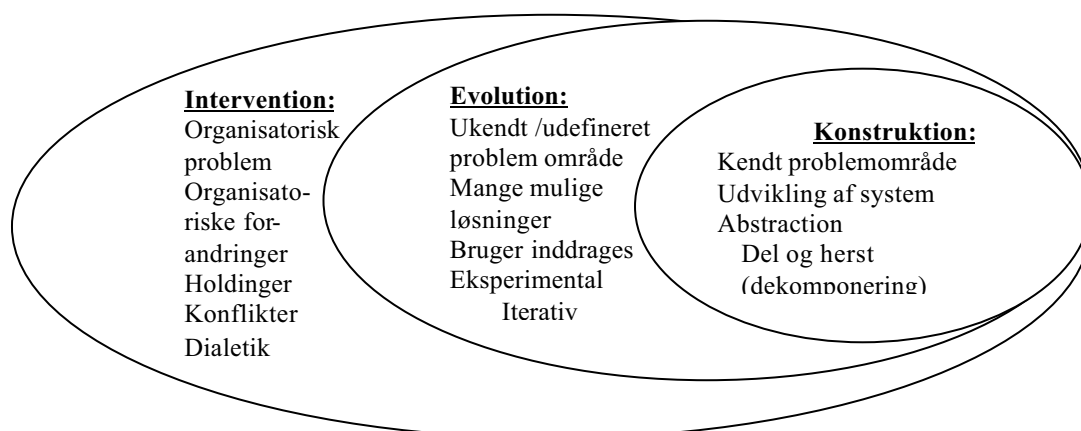
Traditionelle systemudviklingsmetoder, der bygger på konstruktionsparadigmet (fx vandfaldsmodellen), bruger meget tid på at analysere problemområdet. Problemet løses derefter ved abstraktion og dekomponering (del og hersk).

Modellen ligger op til længere systemudviklingsforløb, som ender med en levering af systemet. Dette indebærer to store risici:

- 1) Der går lang tid, fra kravspecifikationen er skrevet, til systemet bliver leveret. Kundens behov kan nemt være ændret i mellemtiden.
- 2) Kunden får ikke mulighed for at lære undervejs; måske var kravspecifikationen uhensigtsmæssig på nogle punkter.

I XP, der bygger på evolutionsparadigmet, antages det, at problemområdet ikke er veldefineret ved projektets start, men at kravspecifikationen bliver ændret som følge af dels bedre indsigt i problemområdet og dels at kundens behov kan ændre sig over tid. Brugeren inddrages i udviklingsprocessen ved fysisk at sætte ham sammen med udviklerne, dette gøres netop for at sammen lære undervejs. Desuden vil et XP projekt typisk indeholde flere leveringer; man starter med en levering af den højst prioriterede funktionalitet, og efterlevere derefter dele med lavere prioritet samt evt. rettelser og tilføjelser. Opdelingen i delleveringer sker ikke for at konstruere systemet vha. nedbrydning, men for at give kunden mulighed for at komme med ændringsforslag undervejs.

XP læner sig ikke op af det sidste store systemudviklingsparadigme; intervention. Interventionsparadigmet ser systemudvikling, som en forandring af brugerorganisationen; det er altså ikke kun et system, som skal udvikles, men hele organisationen, hvor et nyt it-system kan være en del af forandringen. Udviklingen indebærer analyse og forandring af hele organisationen (både den formelle og reelle side), herunder holdninger og konflikter.



**Figur 2 Kontruktion, evolution og intervention**

Man kan vælge at se de tre systemudviklingsparadigmer, som forskellige afgrænsninger for hvad systemudvikling er. Historisk set er paradigmerne opstået i følgende rækkefølge: Konstruktion, evolution og intervention. Men dermed ikke sagt, at det ene paradigme er bedre end de andre; Det afhænger udelukkende af situationen.

I stedet kan man prøve at se på hvilke problemer, som typisk findes ved systemudviklingsprojekter i dag. I Tabel 1 opstilles følgende typiske problemer med softwareudvikling: Overskredne deadlines, projektet bliver lukket, systemet sander til, for fejlbehæftet, misforståede behov og ændrede behov.

En stor del af disse problemer skyldes, at man har set problemområdet som veldefineret og stabilt. Med andre ord; at man har regnet med at kunne løse problemet vha. konstruktion.

Ved i stedet at anvende en evolutionær systemudviklingsmetoder, som XP, vil man i stedet minimere den risikoen for at et projekt mislykkes. Dette sker ved at kunden får en langt højere føling med projektet, ved han deltager i udviklingen. Kunden vil ligeledes have langt højere mulighed for at styre projektets retning, ved at kunne evaluere del-leverancer (det forventes ligefrem at han vil komme med nye og ændrede krav til systemet). Denne øgede kvalitet af risiko-styring giver kunden en bedre muligheder for at styre projektets resultat.

## **Systemudviklingsmetode som kvalitetskontrol**

Traditionelle systemudviklingsmetoder indeholder altid en mere eller mindre eksplicit styring af kvalitet. Analysedokumenter, designdokumenter, forskellige diagrammer, kode-dokumentation, kode-review, test-programmer og bruger-tests kan alle ses som elementer, hvis formål er, at sikre en høj kvalitet af systemet. Hvis man betragter en systemudviklingsmetode som en kvalitetskontrol, kan man derved vurdere hvor god en styring af kvaliteten, som er indeholdt i metoden.

### **Kvalitetskontrol model**

For at kunne lave denne vurdering, har jeg brug for at vide hvilke elementer en kvalitetskontrol består af. Til dette bruges den model, som er beskrevet i CiC<sup>v</sup>, som foreskriver at en kvalitetskontrol basalt indeholder 3 aktiviteter:

1. Planlægning for kvalitet: De ønskede kvaliteter ved hvert artefakt specificeres, herunder hvordan disse egenskaber måles. Disse specifikationer organiseres derefter som forskellige tilstande af produktionen.
2. Produktion: Under selve produktionen kan man forbedre kvaliteten af hvert artefakt ved anvendelse af forskellige metoder og værktøjer. Desuden vil der løbende følges op på produktionsstatus samt løbende evalueringer af (del)produktet.
3. Evaluering og rettelse: Efter produktion bliver artefaktet evalueret imod specifikationen og evt. fejl og mangler rettes.

Jeg vil bruge modellen til at undersøge, hvordan en systemudviklingsmetode bruges som kvalitetskontrol, for henholdsvis de traditionelle systemudviklingsmetoder og XP.

### **'Planlægning for kvalitet' for de traditionelle systemudviklingsmetoder**

Ofte vil der blive lagt meget stor vægt analyse og design aktiviteterne, hvor systemet defineres, beskrives og modelleres. Dette er helt i trin med konstruktionsparadigmet, hvor problemet først defineres og derefter nedbrydes ved hjælp af dekomponering.

Disse aktiviteter kan man vælge at se som en 'planlægning for kvalitet'; ved at en plan (analysedokumenter og designdokumenter) vil man kunne gennemføre produktionen (implementeringen) på en struktureret og forudsigelig måde og derved sikre en høj kvalitet.

### **'Produktion' for de traditionelle systemudviklingsmetoder**

Selve implementeringen sker derefter på baggrund af resultatet fra analyse og designfasen.

Hvordan selve implementeringen sker vil ofte slet ikke være beskrevet eller kun beskrevet meget løst i systemudviklingsmetoden.

### **'Evaluering og rettelse' for de traditionelle systemudviklingsmetoder**

Når selve implementeringen af systemet er fundet sted, så vil test-fasen komme. Man vil ofte lave forskellige 'kvalitetsmålinger', ved hjælp af kodereviews, brugertests. Desuden vil man evaluere ens analyse og design dokumenter, for at sikre at der er konsistens mellem det man havde planlagt at lave og det man rent faktisk lavede.

### **'Planlægning for kvalitet' i XP**

*Metafor:* Giv al udvikling en fælles retning med en fælles historie om, hvordan hele systemet virker. Metaforen beskriver det overordnede mål for projektet.

*Planning game:* En løbende dialog mellem kunden og softwareudviklerne. Kunden får til ansvar at beslutte omfang af systemet, prioritering af elementer i systemet, release-indhold og releasedatoer. Softwareudviklerne skal beslutte estimater for de forskellige elementer, estimater som det samlede projekt, hvordan processen skal organiseres og detailplanlægningen.

### **'Produktion' i XP**

Produktion i XP indeholder en række aktiviteter / regler til styring af kvalitet.

*Test:* I forbindelse med udviklingen af koden, vil man samtidigt udvikle test-kode (unit-tests). Test-koden sikre at den ændring man har lavet, fungerer som den skal. Test-koden bruges desuden til dels at sikre at fremtidige ændringer ikke ødelægger den eksisterende funktionalitet og dels at beskrive de overordnede sammenhænge i systemet, som en slags dokumentation.

Ved brug af test-kode, vil man hurtigt kunne se konsekvenserne ved en ændring man laver.

Kent Beck mener desuden at brug af unit-tests øger produktiviteten: "Det går også hurtigere at kode og teste samtidigt i stedet for kun at kode. Det var ikke en effekt, jeg havde forudset, men jeg lagde hurtigt mærke til den, og jeg har hørt en masse andre sige det samme".

*Enkelt design:* XP vil altid vælge det mest enkle løsning, som tilfredsstillende kundens behov. Dette giver give den mest enkle kode, som dermed også er let at gennemskue og ændre.

*Dokumentation vha. kode:* I XP vil der slet ikke eksistere analyse- og designdokumentation i samme form som ved traditionelle systemudviklingsmetoder. I stedet for at arbejde sig igennem en analyse og designfase, vil man i stedet lave en ekstrem simple implementering af systemet. Hvis man er meget visuel orienteret, er der dog intet i vejen for at man laver små diagrammer, for at få et overblik over systemets arkitektur – diagrammerne skal bare smides ud, når man har forstået ideen og fået overblikket.

I stedet for den traditionelle analyse- og designdokumentation, som typisk er en række dokumenter og diagrammer, skal al dokumentation ske ved hjælp af kode. Man skal altså ved hjælp af små kode eksempler (i de føromtalt tests) beskrive de grundlæggende sammenhænge i det system som man er ved at udvikle. Dette har den klare fordel, at al

dokumentation har samme form (kode) og er samlet eet sted. Dette gør rettelser i systemet meget hurtigere, da man ikke skal sikre konsistens mellem forskellige repræsentationer af dokumentationen.

En teknologi som XP-projekter gør flittigt brug af, er CASE-værktøjer<sup>vi</sup>, hvor koden kan vises og modelleres visuelt. En ændring i kode vil automatisk medføre ændringer i de tilknyttede diagrammer og en visuel design ændring vil medføre en kodeændring.

*Parprogrammering:* Al produktionskode bliver skrevet af to personer ved én maskine. Den ene sidder og koder, mens den anden reviewer koden. Par dannelsen skifter ofte – man har ingen fast makker.

*Kundetilstedeværelse:* Ved at have repræsentant fra kunden siddende sammen med udviklerne, kan mindre uklarheder hurtigt klarlægges og udviklerne bliver ikke fristet til selv at drage konklusioner.

*Kodestandarder:* Udviklerne skriver al kode i overensstemmelse med et sæt regler, der lægger vægt på kodens formidlende rolle.

### **'Evaluering og rettelse' i XP**

*Refaktoring:* Udviklerne omstrukturerer systemet uden at ændre på dets opfattelse ved at fjerne gentaget kode, forbedre den pædagogiske fremstilling, forenkle eller tilføje fleksibilitet.

### **Vurdering**

Det er let at se den enorme forskel på de traditionelle systemudviklingsmetoder og XP, i hvor fokus ligger. Både hvis man ser på antallet af aktiviteter og omfanget af aktiviteter, så vil de traditionelle systemudviklingsmetoder være have fokus på planlægning for kvalitet, hvor XP har fokus i selve produktionen.

Forklaringen findes i, at de traditionelle systemudviklingsmetoder har det med at være meget dokumentations-tunge i 'planlægning for kvalitet'-fasen. Dette er igen især et problem, hvis man skal ind og foretage ændringer af systemet sent i forløbet. Dette kunne f.eks. være at man stod og manglede en klasse til at repræsentere en kontaktperson og som var associeret med en kundeklasse og skulle bruges af salgsafdelingen. Selve udviklingen af den ny klasse ville være en simpel udvidelse, men ud over at lave selve implementeringen, skal man selvfølgelig også holde sin dokumentation ajour. Dette vil sige at ved brug af f.eks. RUP skal følgende dokumentation revurderes: aktør-liste, use-cases, konceptuel model, klassediagram, system sekvensdiagrammer, samarbejdsdiagrammer, samt andre dokumenter fra analyse og designfasen. Med andre ord, så vil rettelser ofte tage længere tid at dokumentere end at implementere.

XP har en ret kontroversiel løsning til dette problem; I stedet for at have fokus i 'planlægning for kvalitet', som de traditionelle systemudviklingsmetoder har det, så er fokus i XP lagt på selve 'produktionen'.

Men hvis man ser en systemudviklingsmetode som en kvalitetskontrol, hvad er så de artefakter som kvalitetskontrollen kontrollerer? Og hvad er artefakternes kvaliteter?

I de traditionelle systemudviklingsmetoder er artefaktet: Implementeringen af systemet. Kvaliteten som man ønsker at højne vha. systemudviklingsmetoden er: Konstruktionen af et system, hvilket indebærer: Korrekthed, stabilitet, effektivitet, integritet og brugervenlighed.

En systemudviklingen er derfor ifølge de traditionelle systemudviklingsmetoder succesfuldt, hvis man har lavet et system, som løser de krav som findes i kravspecifikationen.

I XP er artefaktet både: Definition og implementeringen af systemet. XP antager at definitionen af systemer er delvist ukendt og at den kan ændres over tid. Kvaliteten af 'definitionen af systemet', som man ønsker at højne er: Finde en definition, som tilfredsstillende kundens behov. Implementeringen af systemet, er magen til de traditionelle systemudviklingsmetoder, men måde hvorpå disse kvalitetsegenskaber nås er ikke ved dekomponering, men i stedet anvendes en mere evolutionær fremgangsmåde.

## 4 Konklusion

Kvalitetsbegreberne i XP er en del mere omfattende end i de traditionelle systemudviklingsmetoder. Dette skyldes at XP er en evolutionær systemudviklingsmetode, som ud over at løse et problem også påtager sig at definere problemet (sammen med kunden).

Det vil med andre ord sige, at XP, som de traditionelle systemudviklingsmetoder fokuserer på en række kvaliteter i det endelige produkt (brugervenlighed, korrekthed, stabilitet, etc.). Men ud over dette, beskæftiger XP sig med at definere problemområdet. Dette giver kunden øget mulighed for at styre projektet undervejs og dermed bedre styring af risikoen ved hele projektet.

Der er ingen tvivl om, at en evolutionær systemudviklingsmetode, som XP, vil være at foretrække til at løse et problem, hvor problemområdet er vanskeligt at definere og/eller ændres over tid. Det vanskelige ligger i at bestemme, om problemområdet er fuldstændigt veldefineret og stabilt. Dette har jeg ikke givet noget svar på i dette essay.

## 5 Litteratur

[Kent2000] Beck, Kent: "Introduktion til Extreme programming", IDG 2002 (Oversat fra "Extreme Programming Explained: Embrace Change" 2000)

[Dahlbom1993] Dahlbom, Bo ; Mathiassen, Lars: "Computers in Context – The philosophy and Practice of Systems Design"

## 6 Noter

---

<sup>i</sup> Med de traditionelle systemudviklingsmetoder, tænker jeg på modeller som grundlæggende har rødder i vandfaldsmodellen.

<sup>ii</sup> [Dahlbom1993] side 156

<sup>iii</sup> Redesign [...]. Værktøjerne er designede til at understøtte ændringer i systemet.

<sup>iv</sup> Alternativt kunne man vælge at betragte XP som nogle retningslinier, som man skulle overholde ved systemudvikling. Problemet er bare at mange af reglerne fra XP er uforenelige med de traditionelle systemudviklingsmetoder.

<sup>v</sup> [Dahlbom1993] side 140

<sup>vi</sup> Computer Aided Software Engineering: Programmer som Rational Rose og TogetherSoft ControlCenter