Forelesning 2: Systemutviklingsprosesser

Introduksjon

Å drive utvikling av programvare er en svært utfordrende aktivitet. For å ha en rimelig mulighet til å lykkes viser det seg at teknologi spiller en mindre viktig rolle. Derimot viser det seg at ledelse, organisering, og gjennomføring er viktigere faktorer. I forelesning nr 2 skal vi se nærmere på disse faktorene som samlet faller inn under det vi kaller utviklingsprosesser. I forelesning nr 3 kommer vi inn på prosjektledelse som beskriver hvordan vi kan styre arbeidet i en *systemutviklingsprosess* (heretter kalt bare *utviklingsprosess*). Utviklingsprosess henger tett sammen med *prosjektledelse*, men her vil vi betrakte utviklingsprosess som et overbygg til det hele, mens prosjektledelse er en av de viktige disiplinene i en utviklingsprosess.

Hva er en utviklingsprosess?

En utviklingsprosess beskriver de fundamentale prinsippene for hvordan arbeidet med å utvikle programvaren skal foregå. Det innebærer å beskrive hvilke overordnede faser arbeidet gjennomgår, rekkefølgen på disse, hvilke prinsipper som benyttes for å styre arbeidet, hvilke aktiviteter som skal gjennomføres, hvordan de ulike deltagerne skal organiseres (roller og grupper), hvilke metoder som skal brukes, samt hvilke produkter (artefakter) som skal leveres (dokumenter, programvare, osv.). Dette settes sammen med en prosessflyt som beskriver forløpet på et overordnet nivå.

Den minste enheten av arbeid som skal utføres kalles gjerne en *oppgave*. Dette kan være noe så enkelt som å arrangere et møte, skrive et dokument, eller programmere en del av et større system. I de fleste sammenhenger, vil antall oppgaver bli så stort at vi er nødt til å innføre nye abstraksjonsnivåer (dvs. større enheter) for å klare å holde oversikten. Vi grupperer derfor oppgaver i *aktiviteter.* Ved å navngi aktivitetene, kan vi snakke om dem og dermed håndtere mange relaterte oppgaver på en gang. I mange sammenhenger er det behov for ytterligere abstraksjonsnivåer men det kommer vi ikke til å se nærmere på her.

I litteraturen benyttes begreper som utviklingsprosess, utviklingsmodell, systemutviklingsprosess, og liknende om samme ting. Modell-aspektet er basert på at en utviklingsprosess er en abstraksjon og en forenkling av virkeligheten der vi fremhever de viktige delene og ignorer andre for å fremme forståelsen.

I litteraturen er det beskrevet en rekke ulike utviklingsprosesser, noen er svært abstrakte mens andre er mer konkrete. Det er derfor nyttig å snakke om paradigmer eller klasser av utviklingsprosesser. Dette er altså ikke et nytt abstraksjonsnivå som skal hjelpe oss til å holde oversikten over arbeidet, men en klassifisering som er nyttig for å sammenlikne ulike utviklingsprosesser. En klasse av utviklingsprosesser er basert på de samme fundamentale prinsippene men avviker i gjerne i utformingen, det vil si hvordan de skal gjennomføres. I litteraturen omtales disse klassene på et abstrakt nivå der fasene og rekkefølgen på disse beskriver det overordnede prinsippet. De sier normalt lite om hvilke aktiviteter og oppgaver som skal utføres.

Vi sier gjerne at en navngitt utviklingsprosess er en *instans* av en gitt klasse. Vi kommer ikke til å videreføre klasse som begrep og det vil fremgå av sammenhengen om vi ser på en instans eller en klasse.

Hvorfor er utviklingsprosesser viktig?

En stadig økende konkurranse og frie markeder øver et kontinuerlig press på alle aktører ved å stadig kreve lavere produksjonskostnader og bedre kvalitet. Dette er like relevant i utvikling av programvare som det er i andre kommersielle aktiviteter. Betydningen av dette kan illustreres ved å se på en annen meget stor virksomhet som har hatt en enorm betydning for moderne industri – produksjon av biler. En av de kjente aktørene her er Henry Ford som bygde sin første bensindrevne bil i 1896. I 1903 ble Henry Ford Company opprettet og 5 år senere var Model T klar for markedet. Fabrikk-lokalet bestod av en rekke stasjoner der hver bil ble bygget opp. Bilen stod stille mens alle delene ble fraktet til bilen. I 1913 ble samlebåndet introdusert i Fords fabrikks noe som skapte en revolusjon i bilindustrien. Tiden som medgikk til å produsere en bil ble redusert fra 12,5 til 1,3 timer. Prisen fra 1908 på $825 kunne reduseres og i 1916 ble en Model T solgt for $360. I 1918 var halvparten av alle biler i USA en Ford. Henry Ford hadde skaffet seg et godt forsprang på sine konkurrenter. De smarteste innså at de ikke kunne overleve uten å gjøre det samme som Ford. De andre gikk ganske enkelt konkurs og i 1930 benyttet alle bilfabrikker samlebånd. Betydningen av prosess i dette eksemplet kan enkelt overføres til andre industrielle disipliner, for eksempel utvikling av programvare. Forskjellene i hvordan arbeidet utføres og organiseres kan ha stor påvirkning på produksjonskostnadene og kvalitet. Det betyr også at noen prosesser kan være bedre enn andre. Her må man imidlertid være påpasselig med å presisere hva som menes med ”bedre”.

Hvis vi forlater det kommersielle perspektivet og ser nærmere på hvordan det vil være for den enkelte aktør som arbeider med utvikling, kan utviklingsprosessen fortsatt ha stor betydning. En organisasjon som er opptatt av å redusere kostnader og forbedre kvalitet er normalt avhengig av et veldefinert utgangspunkt og en felles forståelse for hvordan utviklingsarbeidet skal gjennomføres. Å følge en bestemt utviklingsprosess betyr at samtlige aktører kan bli enige om en rekke felles begreper og på denne måten skaper et språk som kan brukes til å etablere en felles forståelse for hvordan arbeidet skal organiseres. Dette er viktige faktorer som påvirker trivsel, motivasjon, og læring. Et felles språk og forståelse er nærmest en nødvendighet for et godt samarbeid. Et annet viktig aspekt ved prosesser generelt er at de er laget for å kunne gjentas.

Ved å etablere felles begreper, og dokumentere utviklingsprosessen, vil organisasjonen skape en basis og en praksis som til enhver tid reflekterer organisasjonens beste tilnærming til sin produksjon av programvare. Dette gir organisasjonen mulighet til å bruke sin beste kunnskap på hvert eneste utviklingsprosjekt. Denne formen for repetisjon er ikke bare viktig for å utføre hvert prosjekt på best mulig måte, men den skaper også et nødvendig utgangspunkt for forbedring. For i det hele tatt å kunne snakke om forbedringer og spre disse i en organisasjon må man ha et fast utgangspunkt. Utviklingsprosesser er derfor et nødvendig verktøy for å etablere et slikt utgangspunkt.

Forskning på et stort antall utviklingsprosjekter viser at rett valg av utviklingsprosess kan gi organisasjonen en rekke fordeler. Dessverre viser også denne forskningen hvor vanskelig det er å lykkes. En undersøkelse utført av Simula i 2003 viser at forutsigbarheten i utviklingsprosjekter er svært lav. Av de prosjektene som ble studert overskred 76% budsjettet. Bare 19% av prosjektene klarte å levere under budsjett. Den gjennomsnittlige overskridelsen var på hele 41%. Det viste seg at

valg av utviklingsprosess har stor betydning for hvor stor andel av prosjektene som hadde overskridelser. Ved bruk av den såkalte fossefallsmodellen (denne kommer vi tilbake til senere) var overskridelsene på hele 55%, mens med iterative/inkrementelle/evolusjonære prosesser lå overskridelsene på 24%. Andre undersøkelser utført gjennom ulike forskningsprosjekter indikerer også at en stor andel av utviklingsprosjektene aldri fullføres. Også her spiller valg av utviklingsprosessen en vesentlig rolle. En viktig observasjon i denne sammenheng er de nedslående resultatene fra disse undersøkelsene. Svært få prosjekter lykkes i å levere en løsning innenfor de fastsatte tids- og kostnadsrammer. Det betyr igjen at sannsynligheten for å lykkes er langt lavere enn vi kunne ønske. Det er derfor svært overraskende å observere optimismen som råder i de ulike miljøene i programvare-industrien.

Livssyklusmodeller

En livssyklusmodell beskriver en utviklingsmodell som tar for seg forløpet fra unnfangelsen av en ide om å utvikle et system til realisering, drift, vedlikehold, og nedleggelse. Det finnes ingen enhetlig og presis definisjon av hvilke faser en slik modell inneholder. Det viktigste er å forstå hvilke hovedtrinn en livssyklusmodell bør inneholder:

1. Kravinnsamling og kravanalyse
2. Design
3. Programmering
4. Test
5. Drift og vedlikehold

Merk at dette er en forenkling, og hensikten er å hjelpe oss med forståelse og kommunikasjon. For eksempel må man rimeligvis gjennomføre testing i drift og vedlikehold. En livssyklusmodell blir derfor gjerne en idealisert forenkling av virkeligheten.

Kravinnsamling og kravanalyse

Formålet med kravinnsamling og kravanalyse er å definere hvilke problemer systemet skal løse og på hvilken måte dette skal skje.

Normalt bør resultatet av slike analyser inneholde en overordnet målsetting som uttrykker hvorfor vi ønsker å utvikle dette systemet og hva vi ønsker å oppnå. Før vi starter å utvikle et nytt tekstredigerings-system, bør vi ha det klart for oss hvorvidt vi kan lage et slikt system som er vesentlig bedre enn eksisterende systemer, hvilken ny funksjonalitet vi vil tilby, om markedet vil finne et nytt slikt system verdt å kjøpe, osv. Dette er en svært viktig del av systemutviklingen som dessverre ofte blir ignorert. Formålet med å definere dette er at det skal benyttes i de mange beslutningene som må tas underveis i utviklingen. For å kunne definere formålet så presist som mulig er det også nyttig å beskrive noen av de viktigste begrensingene i målsettingen. Ved å si noe om hva vi ønsker å oppnå og samtidig sette en grense for hvor langt vi er villig til å gå, kan vi få en rimelig god beskrivelse som kan benyttes for videre arbeid med systemet.

Når vi har en klar definisjon av målsettingen og begrensingene, kan vi begynne på den virkelige jobben med å definere de enkelte krav. Disse definerer alle de funksjoner systemet skal kunne tilby brukeren. Skal vi for eksempel utvikle et tekstredigerings-system bør vi kunne få lov til å skrive inn noe tekst, flytte den rundt, slette, etc. Dette kalles *funksjonelle* krav. De funksjonelle kravene definerer kun hvordan et system skal brukes, de sier ikke noe om de tekniske aspektene. Skal vi for

eksempel lage et billettsystem som er tilgjengelig på internett er vi normalt interessert i å si noe om hvor mange samtidige treff og hvor mange samtidige brukere systemet må kunne håndtere uten å stoppe. Dette kalles *tekniske* krav eller *ikke-funksjonelle* krav. Legg merke til at kravene ikke uttrykker noe om hvordan systemet skal bygges verken om gjennomføring eller oppbygning. Kravene skal kun si noe om hva som skal oppnås. En annen måte å se dette på er at vi søker å besvare spørsmål av typen hva… og ikke hvordan…

Det viser seg at *hvem* som definerer kravene kan ha stor påvirkning på om utviklingsprosjektet lykkes eller ikke. Frihetsgradene i systemutvikling er store og det er svært utfordrende å definere kravene slik at det endelige systemet løser de intenderte problemene. Hvem som definerer kravene blir derfor sentralt. Alle personer som på en eller annen måte bør delta i å definere kravene kalles interessenter (også kalt ”aktører”, og på engelsk: ”stakeholders”). I enkelte tilfeller, defineres også tekniske installasjoner som interessenter, men vi skal holde oss til personer. Interessentene kan med fordel inndeles i grupper avhengig av hvilket perspektiv de har. Normalt vurderes sluttbruker (også kalt bare ”bruker”) som den viktigste gruppen. Dette er de personene som skal bruke systemet til å løse en oppgave. En sluttbruker kan være en person som bestiller en billett over internett eller en saksbehandler i offentlig forvaltning. Dersom systemet er så vanskelig å forstå eller navigere at halvparten av alle som ønsker å kjøpe en billett ikke finner frem til billetten de ønsker, har vi åpenbart feilet. Andre viktige grupper av interessenter er driftspersonell, systemeier (den eller de som er ansvarlig for forvaltningen av systemet), personer som jobber med brukerstøtte, eller konsulenter som jobber med å utvikle spesielle tilpasninger.

Arbeidet med kravene dokumenteres normalt i et dokument og kalles kravspesifikasjon. Størrelsen på slike dokumenter varierer fra noen få sider til flere tusen i store prosjekter. De ulike typene utviklingsprosess stiller også ulike krav til omfanget av dokumentene. Enkelte typer prosess eller instanser av slike stiller svært strenge og detaljerte krav til krav dokumentene og prosessen som skal følges fra de utarbeides til godkjenning (som kan være en omstendelig prosess). Legg merke til at feil som gjøres på dette stadiet kan ha fatale konsekvenser for slutt-resultatet. Feil beslutning om hvilke funksjoner som systemet skal ha eller feil forståelse av problemet systemet skal løse kan føre til at alt etterfølgende arbeid vil fokusere på å utvikle et system som ikke kan brukes til den tiltenkte oppgaven. Det er omtrent som å begynne å utvikle en motorsykkel når det vi egentlig har behov for er en lastebil.

Designfasen

Gitt målsettinger og krav og en forståelse for hva som skal oppnås, kan vi begynne å se på hvordan systemet skal bygges opp. Nå snakker vi ikke om gjennomføringen men om selve konstruksjonen. Denne fasen kalles designfasen. I designfasen skal vi for eksempel beslutte hvilke skjermbilder vi må lage og hvordan de skal se ut. Videre må vi bestemme oss for hvordan arkitekturen i systemet skal være. Det innebærer å identifisere hovedkomponentene i systemet og bestemme hvilket ansvar hver enkelt komponent skal ha. Videre må vi bestemme hvilken relasjon det skal være mellom de ulike delene og hvordan de skal samarbeide for å bidra til at vi oppnår målsettingene. I mange tilfeller benyttes egne modelleringsspråk for dette arbeidet, og design beskrives i egne diagrammer bygget opp ved hjelp av en egnet notasjon. I tilegg til slike diagrammer, dokumenteres ofte resultatet i det vi kaller en systemspesifikasjon.

Hvis vi gjør en dårlig jobb i denne fasen risikerer vi at arbeidet med å utvikle systemet blir uforholdsmessig komplisert og dyrt. Senere kan vi få problemer med vedlikehold ved å pådra oss uforholdsmessige høye kostnader og dårlig kvalitet. For tiltrekkelig store og kompliserte systemer vil også levetiden reduseres betraktelig og de totale levetidskostnader vil da være større enn nødvendig.

Programmering

Etter at vi har definert målsettinger, krav, og design kan vi begynne på programmeringen. Det er først nå at vi begynner å utvikle sluttproduktet. Denne fasen inkluderer ytterligere forfining av krav og design.

Testfasen

Etter at systemet er utviklet må vi forsikre oss om at det oppfyller målsettingene og kravene som ble definert under kravinnsamlingen. For å gjøre dette må systemet testes grundig. Dette kalles testfasen. Det viktigste vi må forsikre oss om, er at vi har utviklet en riktig løsning som løser de problemene vi ønsker å løse med systemet. En viktig del av testing blir derfor å etterprøve at våre opprinnelige målsettinger var korrekte og at kravene faktisk var riktige, samt at kravene faktisk er oppnådd. Det er rimeligvis svært sent å studere dette etter at systemet er ferdig utviklet, men vår forståelse av de problemstillingene vi ønsker å løse forandres underveis og vi blir ofte tvunget til å endre oppfatning om hva som er rett løsning. Senere skal vi komme tilbake til hvordan vi kan gjøre slike vurderinger på et tidligere stadium i prosessen. At systemet løser de oppgaver som vi ønsker, er imidlertid ikke tilstrekkelig. Systemet må også gjøre dette på en effektiv måte slik at vi bidrar til lavere kostnader eller bedre kvalitet og presisjon i arbeidet som utføres i den organisasjonen som skal bruke systemet. Dette innebærer at vi må undersøke om systemet er effektivt i bruk, at det er lett å navigere i skjermbildene, lett å lære, etc.

Videre må vi kontrollere at systemets tekniske egenskaper er i henhold til behovene og kravene, og at det er tilstrekkelig fritt for feil.

Målsettingen med testfasen er å finne flest mulig feil. Dette er viktig fordi vi bør vite hvilken risiko vi pådrar oss på det tidspunktet vi ønsker å ta systemet i bruk. Vi kan aldri bruke testing til å vise at systemet er fritt for feil. Testing vil kunne vise oss noen av de feilene vi har, hvordan de er distribuert i programkoden og hvor ofte de inntreffer. Dette er imidlertid svært viktig informasjon og desto bedre vi kjenner systemet, desto bedre vil vi kjenne risikoen ved å sette det i drift.

Faseinndelingen som vi nå har sett på, er en svært idealisert og forenklet fremstilling av hvordan programvare utvikles. De klare skillene og rekkefølgene er først og fremst satt opp slik for å beskrive en logisk oppdeling av forløpet fra ide til systemet er ferdig testet og klart for levering. At det her fremstilles som faser i en kronologisk rekkefølge er gjort for å bidra til en overordnet forståelse av hvilke hovedaktiviteter som inngår. Senere skal vi se nærmere på hvordan vi kan endre på rekkefølgene for å oppnå bedre produktivitet og kvalitet.

Klasser av utviklingsprosesser

Innledningsvis ble det nevnt at vi kan snakke om ulike klasser av utviklingsprosesser. Dette er altså ikke konkrete navngitte prosesser, men mer en kategorisering. Det finnes mange måter å gjøre denne inndelingen på, og her har vi valgt en forholdsvis enkel tilnærming. Vi deler derfor utviklingsprosessene inn i fire ulike klasser:

1. Prøv-og-feil
2. Fossefallsmodellen
3. Prototyping
4. Evolusjonær, iterativ, og inkrementelle modeller

Innenfor hver av disse klassene finnes det igjen underklasser som oppfyller de viktigste prinsippene men som har sine egne varianter når vi studerer dem på nærmere hold. I det følgende skal vi gå igjennom hver av disse modellene.

Prøv og feil

Prøv-og-feil fortjener nærmest ingen egen klasse men det er allikevel viktig å kjenne til denne for å gjenkjenne en praksis som vi normalt ønsker å unngå. Denne prosessen kan knapt kalles en prosess fordi den ikke er repeterbar, men nærmest en ad-hoc tilnærming der alt er lov. Det er imidlertid mulig å dele den opp i to sentrale trinn, programmering og feilretting. Den inneholder ingen strukturert planlegging, og hvorvidt det utarbeides en kravspesifikasjon eller ikke, er tilfeldig og varierer innenfor samme prosjekt. Det samme gjelder design. Det betyr at det er opptil den enkelte deltager om dette gjøres eller ikke. Prøv-og-feil er den minste prosessen som ethvert prosjekt normalt må igjennom for overhodet å kunne levere noe som helst. Uten programmering vil ingenting realiseres, og av erfaring vet vi at uten feilretting vil viktige deler av systemet ikke virke.

Fossefallsmodellen

Fossefallsmodellen er det nærmest vi kommer en kronologisk sammensetning av de ulike livssyklusfasene over. Den starter gjerne med en foranalyse der selve grunnlaget for prosjektet studeres, fortsetter med kravinnsamling, design, programmering og test. I noen tilfeller legges også prosjektstyring og andre mer varige delprosesser inn. Dette er prosesser som varer fra start til slutt.

Det finnes mange varianter av fossefallsmodellen og de fleste av disse som finnes i litteraturen er gjerne mer detaljert. Formålet her er i hovedsak å presentere en enkel variant som viser de overordnede prinsippene og mønsteret slik at det er mulig å gjenkjenne og diskutere modellen.

Det mest sentrale prinsippet bak fossefallsmodellen er at den baserer seg på at utvikling av programvare er en forutsigbar produksjonsprosess. Det betyr blant annet at vi kan etablere en pålitelig og detaljert plan ved oppstart. Fra kravene og gjennom designfasene vet vi hvilket system vi skal utvikle og hvordan det skal bygges opp. Denne informasjonen dokumenteres og godkjennes av interessentene. Underforstått i fossefallsmodellen ligger en antagelse om at vi faktisk er i stand til å gi en tilstrekkelig beskrivelse av systemet slik at interessentene kan godkjenne løsningen før utviklingen starter. I en streng tolkning av fossefallsmodellen skal hver fase avsluttes før neste kan påbegynnes. Videre ligger det også en antagelse om at vi ikke forventer endringer underveis i utviklingen etter at fasene er avsluttet og godkjent. Merk at det finnes ulike varianter av fossefallsmodellen med mulighet for å gjenta en fase dersom dette er nødvendig. Dette er mer en støtte for å håndtere eventuelle avvik i fossefallsmetoden, enn en ny type prosess. Videre innholder fossefallsmodellen en antagelse om at vi vil klare å gjennomføre utviklingen på en tilfredsstillende måte på første forsøk. Når testfasen starter må vi anta at vi i tilstrekkelig stor grad har bygget den riktige løsningen og at testfasen i hovedsak skal benyttes til å avdekke feil og å ha en rimelig god forståelse av hvilken risiko vi løper ved å settes systemet i drift.

Fordelen med fossefallsmodellen er at den er konseptuelt enkel med tydelig start og stopp for planlagte aktiviteter. Det er enkelt å relatere modellen til den tradisjonelle tankegangen innen produksjon for øvrig. En annen viktig egenskap er at dersom evalueringen av krav og design kan gjøres med tilstrekkelig sikkerhet før programmeringen starter vil dette bidra til at feil og mangler oppdages tidlig i prosessen, noe som har en svært positiv konsekvens for kostnadene knyttet til å utbedre disse problemene.

En av ulempene med fossefallsmodellen er at det sjelden er mulig å gjøre en god evaluering av systemet kun basert på krav- og designdokumenter. Det viser seg at det er først når interessentene kan prøve systemet (eller deler av systemet) at de er i stand til å gjøre en tilstrekkelig evaluering.

I dette ligger det problem at de som skal beskrive kravene og forslag til design ikke klarer å se for seg en tilstrekkelig god løsning på forhånd. Så vi risikerer ikke bare at kravene er feil men også at vesentlige krav mangler. Interessentene vet ganske enkelt ikke hvordan systemet bør fungere før de har sett ett eller flere forslag til løsning.

Et annet problem er at fossefalsmodellen i streng forstand ikke gir god støtte for å gjøre endringer etter at krav og design er godkjent. Dette har blitt løst ved å tillate å gå tilbake og gjenta tidligere faser.

Legg også merke til at når testing utføres til slutt i prosjektet er det svært vanskelig på forhånd å estimere hvor mye arbeid som er påkrevet for å identifisere tilstrekkelig mange feil og mangler, samt å rette disse. Vi har ingen empiri eller historiske data fra dette prosjektet som kan hjelpe oss med å planlegge test og feilretting.

I praksis betyr dette at ved å følge en fossefallsmodell risikerer vi at vesentlige feil og mangler først blir oppdaget i testprosessen. Med vesentlig menes her problemer som ikke kan utbedres på en enkel måte uten at det påvirker prosjektets forløp. Vi pådrar oss derfor en risiko for å overskride budsjett eller forsinke planlagt leveranse.

En annen ulempe er at fossefallsmodellen krever forholdsvis omfattende mengde med dokumenter for å kunne gi en beskrivelse som er tilstrekelig for å kunne vurdere om løsningen er rett eller ikke.

Prototyping

I stedet for å forsøke å beskrive hvordan et system vil fungere i dokumenter, kan man lage en prototype som er en forenklet versjon av systemet som fungerer som et forslag til det endelige produktet. Det finnes ulike tilnærmingen til dette, fra å lage såkalte papirprototyper til mer realistiske enkle systemer som demonstrerer de viktigste funksjonene i systemet. I kontekst av utviklingsprosesser er oppmerksomheten i hovedsak rettet mot protoyper av den siste typen. Prototypen kan enten kastes etter bruk når man har funnet en tilfredsstillende løsning, eller den kan brukes som utgangspunkt for videre utvikling til det endelige sluttproduktet. Prototyper kan gi oss en mer levende fremstilling av hva vi foreslår å utvikle og på den måten gi et mer realistisk inntrykk enn hva vi får gjennom å lese dokumentasjonen. Ulempen er at uten god disiplin kan prototypen lett benyttes som grunnlag for videre utvikling, uten at det kanskje var den egentlige hensikten. Visuelt og funksjonelt kan dette være et godt grunnlag men vi risikerer at utgangspunktet vårt ikke er godt nok som en teknisk plattform for videre utvikling.

Evolusjonær, iterativ, og inkrementelle utviklingsprosesser

Den siste og mest sentrale klassen av utviklingsmodeller er de som bygger på en evolusjonær tilnærming – også ordene ”iterativ” og ”inkrementell” blir brukt i denne sammenheng. For enkelthets skyld kaller vi disse for evolusjonære utviklingsprosesser selv om dette ikke er helt presist og korrekt. De evolusjonære prosessene søker å unngå noen av de viktigste ulempene med de øvrige prosessene. Til en viss grad kan vi si at de inkluderer noen aspekter av fossefall og prototyping, dog har nok ideen om prototyping blitt sterkere fremhevet i de mest moderne metodene mens den trinnvise tilnærmingen i fossefall har gradvis forsvunnet.

Det finnes ingen entydig og klar definisjon på begrepene evolusjonær, iterativ og inkrementell, og vi har i denne sammenheng valgt en definisjon som binder lærestoffet sammen.

En enkel måte å beskrive en iterativ tilnærming er å dele et prosjekt inn i flere og mindre mini-prosjekter (iterasjoner) der hvert mini-prosjekt fungerer i henhold til fossefallsmodellen. De fleste metodene har gått bort i fra at hver iterasjon skal følge fossefallsmodellen men det kan være en enkel måte å se sammenhengene på. I praksis vil man normalt redusere krav til dokumentasjon og en del av de øvrige formalistiske og lite praktiske delene av fossefallsmodellen.

Formålet med evolusjonære modeller er å støtte en praksis der planer og løsninger forbedres i hver iterasjon. Etter at en iterasjon er avsluttet kan resultatet evalueres og dersom det ikke er tilfredsstillende kan man forsøke på nytt. Da er det viktig at den delen som ikke oppfylte behovene er tilstrekkelig liten slik at kostnadene forbundet med å utvikle den på nytt ikke blir for store. En viktig målsetting er å oppdage feil (i krav, planer, løsning) så tidlig som mulig og få muligheten til å rette det opp nærmest umiddelbart.