Studentnummer:154542

Universitetet i Bergen

Institutt for informasjons- og medievitenskap

INFO 214

Systemutvikling

Høst 2010

Fagansvarlig:

Ankica Babic

Innleveringsfrist er 3. desember 2010

Oppgaven leveres elektronisk via Studentportalen

innen klokken 13:55

Smidig systemutvikling

Innledning

Å utvikle et informasjonssystem er en komplisert prosess, det krever godt forarbeid, det krever kunnskap om ulike verktøy, det krever kommunikasjon på et høyt gjensidig nivå mellom utvikler og en eventuell kunde.. Et informasjonssystem er et system som står for innsamling, representasjon, transformasjon, kommunikasjon og presentasjon av informasjon(Avison et al.2006)

Når et system blir laget vil det ta for seg en del av et univers, det kan være alt fra å holde kontroll på alle utgifter til en bedrift, til et sosialt medium, og videre til å holde kontroll på den neste NASA raketten som skal opp i verdensrommet. Disse tre systemene vil ha ulike krav, de vil også ha ulike bruksområder og ikke minst ulike kriterier for hvordan de skal utvikles. Denne teksten handler om det agile perspektivet innfor systemutvikling, og vil beskrive hvordan systemutviklingsmetoder har utviklet seg i takt med informasjonsteknologien fra 1960 tallet og opp til tidlig 2000 tallet. Første avsnittet forklarer kort om programvareutvikling sett i sammenheng med hvordan informasjonssystemene utviklet seg og hvilke funksjoner de skulle støtte opp under. Det andre avsnittet handler om den agile alliansen og essensen i den smidige systemutviklingen, samt alliansen sine motstandsargumenter for å bryte med de tradisjonelle modellene. Det siste avsnittet tar opp kritikk mot agile metoder og oppsummerer samt en konklusjon.

Fagfeltet software engineering

De tidlige applikasjonene for datamaskiner opp til 1960 tallet, ble stort sett implementert uten at det eksisterte noen eksplisitt systemutviklingsmetode. I disse tidlige dagene lå tyngde punktet innefor programmeringsområde, og ferdighetene til programmereren var høyt verdsatt. Systemutviklere hadde høye tekniske ferdigheter men ikke nødvendigvis gode kommunikasjons kunnskaper. Dette betydde ofte at behovet for brukeren i applikasjonsområde ikke var bra nok etablert, og konsekvensen ble ofte at informasjonssystemet sitt design var noen ganger mindre passende for applikasjonen. Få programmere ville følge en formell metode, ganske ofte brukte de tommelfinger regelen og lente seg på erfaring. Estimering av dato på når systemet var operasjonelt var vanskelig å si, og applikasjonene var ofte etter skjema. Programmereren var ofte overarbeidet, og brukte ganske mye av tiden til å rette opp og styrke applikasjonene som var operasjonelle. ”This was not a methodology, it was only an attempt to survive the day ”(Avison et al. 2006 s 23). Etter hvert som datamaskiner ble brukt mer og mer og bedriftene krevde mer passende applikasjoner for de dyre utgiftene sine, kunne ikke denne tilstanden vedvare. Det førte til tre store forandringer:

Den første var en økende forståelse av den delen av utviklingsdelen som gjelder analyse og design og også å verdsette den rolle som systemanalytiker like mye som programmerer rollen.

Den andre var en erkjennelse på at, etter hvert som organisasjoner vokste i størrelse og kompleksitet, var det ønskelig å gå bort fra en et systems løsning og mot et mer integrert informasjonssystem.

Det tredje var et styrket ønske om å bruke aksepterte metodologier for utvikling av informasjonssystemer. (Avison et al.2006)

Tradisjonell systemutvikling/Fossefallsmetoden

Innføringen av livssyklustilnærming SDLC(Information systems development life cycle), hadde en stor innflytelse som en generell tilnærming til å utvikle Informasjons-systemer"(Avison et al. 2006 s 31). Det er mange varianter men de følger denne samme strukturen: feasibility study, system investigation, system analysis, system design, implementation, review and maintanance. Disse stegene sammensatt er ofte referert til som "conventional system analysis", "traditional system analysis", "systems development life cycle", og ganske ofte i USA som "the waterfall modell"(1970, Winston W Royce). Utrykket "life cycle" indikerer den iscenesatt naturen av system utviklingsprosessen.

Fossefallmetoden blir ofte illustrert av en typisk nedbryting av alle stegene, et hvert steg blir lukket og kvalitetssikret før et nytt steg påbegynnes(Avison et al.2006). Fossefallsmodellen har klare mål for hver fase, når en fase er utført vil utviklingen gå videre til neste fase og ikke tilbake. Fordelen med fossefall modellen er at den gir muligheter for å dele opp oppgaver og gi dem til ulike avdelinger og ha ledelsemessig kontroll. En tidsestimering kan settes for hver fase og produktet kan fortsette igjennom utviklingen og bli levert i "tide". Utviklingen går igjennom steg som konsept, design, implementering, testing, installering, feilsøking og ender opp med drift og vedlikehold. Hver fase går igjennom strenge krav uten noe overlapp eller iterative trinn. Ulempen med fossefall modellen er at den ikke tillater mye reflektering eller revisjon, Når en applikasjon er i test fasen er det vanskelig å gå tilbake å endre på noe som ikke var gjennom tenkt nok i konsept fasen (Weaver 2004).

RUP(Rational Unified Process), i 1998 erklærte Jacobson at metode krigen var over. Han mente at en standard var blitt skrevet, og det var Unified Modeling Language(UML). UML er et modellerings språk, å oppdage hva som skal bli modellert er ganske ulikt. For dette har Jacobson og andre(Rumbaugh, Booch, Kruchten, og Royce) videreutviklet og formet en prosess som har blitt kjent som Unified Process og som benytter UML for modellering. RUP er en "arkitektur-sentrisk" prosess. Programvare arkitektur er sammenlignet med arkitekturen av å bygge en bygning eller hus, hvor man får se designet av bygget før det blir bygget. System-arkitekturen er lik på mange måter, men viser ulike syn på systemet. En får beskrivelser av maskinvare, operativ system, databaser og mer. RUP er også beskrevet som iterativ og inkrementell, på grunnlag av at krav til systemet ikke kan avklares i en prosess, utviklingsfasene av systemet vil lede til forståelse om systemet som vil gi forandringer i kravspesifikasjoner. Prosjektets resultat vil komme over tid i form av en serie med kontrollerte overleveringer for å redusere risikoen for at systemet skal feile. (Avison et al.2006)

Evolusjonen av informasjonssystemer

I mellom 1965 og 1975 konsentrerte ledere seg om og automatisere de funksjonene som de kunne, slik at bedriftene ble mer effektiv. Disse inneholdt de typiske prosessene som behandlet mange rutinemessige transaksjoner som lønn, lager status, fakturaer. Avdelingsledere delegert ofte ansvaret for informasjonsstyring til en informasjonssystem avdeling, disse ble flinke til å kjøre store, rutinebaserte og vanligvis sentraliserte systemer. Denne funksjonen ble veldig innflytelsesrik etter hvert som de ledende avdelingene gav informasjonssystem saker videre til spesialister. Denne teknologien eksisterte blant de store aktørene. I det neste ti året spredte automatiserte systemer seg voldsomt. Teknisk utvikling gjorde mindre systemer mulig og mer attraktiv for ledere i andre deler av organisasjonen.

Siden midten av 1980 tallet har teknisk utvikling brakt informasjonssystemer i forgrunnen av bedriftenes policy. Systemer som i tiår har støttet bedriftenes kjernefunksjoner som finans, produksjon og distribusjon, forsetter å utvikle seg å sysselsette mer teknologi. Datamaskin baserte systemer har blitt utvidet til å tjene mange andre foretningsfunksjoner og blir brukt til å støtte bedrifts prosesser i en mer integrert retning. Programvare leverandører har utvidet produkt linjen til å støtte funksjoner så forskjellige som produkt prognoser, leverandør rating og prosjekt ledelse. Informasjonssystemer støtter nå ledere og den profesjonelle staben direkte i de fleste foretningsavgjørelser.

Etter at internett kom på banen rundt midten av 1990 tallet har det stimulert til mer utvikling på disse områdene. Det utfordrer tradisjonelle organisasjoner til å innovere prosessene, samt til å integrere dem mot leverandørene og kunden. Dette leder også til bedriftens transformasjon, gjenoppfinning av verdi kjeden og søke mot nye veier for å gjøre foretninger. Utviklingen av mobiltelefonen har befridd internett fra den klassiske datamaskinen, folk kan nå finne informasjon rettet mot foretningsvirksomhet eller sosiale områder nesten hvor som helst. Mange markeder blir mer gjennomsiktelig og kunder viser helt nye handlemønster og adferd. Mange kommentatorer tolker det som en fundamental forandring fra en industriell økonomi til en mer global nettverks økonomi hvor bedriftene bare kan overleve hvis de er agile, fleksible og adaptive til de nye teknologiene.(Boddy et al.2008)

Den agile alliansen

I 2001, motivert av observasjon av at programvareutviklere(lagene) i mange selskap ble sittende fast i hengemyren av stadig økende prosesser, møttes en gruppe industri eksperter for å skissere de verdier og prinsipper som ville tillate software team til og arbeide raskt og reagere på endringer. I løpet av noen måneder jobbet de sammen for å lage et dokument med disse verdiene og prinsippene. Resultatet var det agile manifestet av den agile alliansen. (Martin et al. 2003)

Det agile manifestet

Vi avdekker bedre måter å utvikle programvare på ved utvikle og hjelpe andre å utvikle. Gjennom arbeidet har vi kommet til verdiene:

- *Individ og samhandling før prosesser og verktøy*
- *Fungerende programvare før omfattende dokumentasjon*
- *Samarbeid med kunden før kontraktsforhandlinger*
- *Å svare på endring før å følge en plan*

Det vil si at selv om det er verdi i punktene på høyre siden så verdsetter den agile alliansen punktene på venstre side mer(Beck et al. 2001).

Individ og samhandling før prosesser og verktøy

Mennesker er den viktigste ingrediensen til suksess. En god prosess vil ikke hjelpe på resultatet hvis laget ikke har sterke spillere og en dårlig prosess kan gjøre den sterkeste spilleren mindre effektiv. Selv en gruppe av sterke spillere kan feile grovt hvis de ikke samarbeider. En sterk spiller er ikke nødvendigvis en super programmerer, en sterk spiller kan være en gjennomsnittsprogrammerer, men en som jobber godt med andre mennesker. Kommunikasjon og interaksjon er mer viktig en rå programmerings talent. Et lag med gjennomsnittsprogrammerere som kommuniserer bra sammen har større mulighet for å lykkes enn en gruppe av superstjerner som ikke klarer å jobbe som et lag. De riktige verktøyene er også viktig for suksess. Kompilatorer, IDEs, kildekode kontrollsystemer m.m. spiller en viktig rolle for de som utvikler, slik at de kan utvikle. Det finnes mange ulike verktøy, det dyreste og siste på markede er ikke nødvendigvis det beste. Verktøyene bør være i takt med menneskene som skal bruke det, det må være verktøyer de behersker og mestrer i praksis.

Fungerende programvare før omfattende dokumentasjon

Programvare uten dokumentasjon er ikke bra. Kode er ikke det ideelle medium for å formidle begrunnelsen og strukturen til et system. Man trenger å produsere dokumenter som er leselig for mennesker som beskriver systemet og grunner til valg av design. For mye informasjon er verre en for lite informasjon, mye programvare dokumentasjon tar tid, og dokumentere tar mer tid hvis det skal utføres i takt med systemets utvikling. De er viktig for utviklere å skrive et kort fattet, rasjonelt og strukturert dokument som kan vedlikeholdes og oppdateres. Dette dokumentet bør inneholde begrunnelse, diskusjon om design og beskrivelse av systemet på et høyt nivå. Når andre utviklere skal jobbe med eller på systemet, så må man jobbe tett sammen med dem. De to faktorene som er de beste til å overføre informasjon til andre utviklere er kildekoden og utvikleren. Koden er fast satt, den kan være vanskelig å forstå og dra mening ut fra, men det er den eneste entydige kilde av informasjon. Utvikler holder veikartet til den evige forandringen i hodet sitt. Det er ikke noe raskere eller mer effektiv måte å overføre veikartet til enn ved menneske til menneske interaksjon.

Samarbeid med kunden før kontraktsforhandlinger

Programvare kan ikke bestilles som en handelsvare. Du kan ikke skrive en beskrivelse på programmet du vil ha for så å få noen til å utvikle det til en oppsatt plan med en oppsatt pris. Gang på gang viser det seg at hvis man behandler programvareprosjekter i denne formen så vil det ikke lykkes. Det er fristende for en leder av en bedrift å si hva han vil ha til utviklerlaget, og forvente at laget trekker seg tilbake for en periode deretter å komme med et fullstendig ønsket system ut i fra kravene. Suksessfulle prosjekter inkluderer tilbakemelding fra kunden på tilfeldig og jevn basis. En kontrakt som spesifiserer kravene, planen og kostnaden av prosjektet er fundamentalt feil. Den beste kontrakten er den som styrer veien utviklerlaget og kunden skal jobbe sammen.

Å svare på endring før å følge en plan

Det er muligheten til å reagere på forandringer som utgjør om programvare prosjekter skal ha suksess eller ikke. Planene må være fleksible og klar til å møte endringer i forretningsvisjon og teknologi. Retningen til programvare prosjekter kan ikke være planlagt langt ut i fremtiden, foretningsmiljøet kan forandre seg, slik at krave endres også. Kundene vil nok også forandre kravene etter at de har sett systemet i bruk. Selv om vi vet kravene og sett at de ikke vil endre seg, vil det være vanskelig å estimere hvor lang tid det vil ta og utvikle dem.(Martin et al.2003)

Prinsipper

Verdiene ovenfor inspirerte til en rekke prinsipper, disse prinsippene er det karakteristiske som skiller et sett med agile praktiser fra tungvekts prosesser, eller tradisjonelle metoder. Prinsippene er som følger:

- Tilfredstile kunde ved og lever verdifullt programvare tidlig og ofte.
- Ønske velkommen forandring ang krav, selv om det er seint i utviklingen.
- Fungerende programvare er levert ofte, fra noen uker til noen måneder.
- Kunde og utvikler må jobbe sammen daglig igjennom prosjektet.
- Prosjekter skal bli bygget rundt motiverte individualister. Hvis det riktige miljøet og støtten er tilstedet så kan utvikleren få tillit til å få jobben gjort.
- Den mest effektive metoden for å videreføre informasjon er via samtale og ansikt til ansikt.
- Fungerende programvare er primær faktoren for å se progresjon.
- Smidige prosesser fremmer bærekraftig utvikling. Sponsorene, utviklere og bruker skal kunne opprettholde et konstant tempo på ubestemt tid.
- Oppmerksomhet til teknisk dyktighet og design øker smidigheten.
- Enkelhet, kunsten å maksimere mengden av arbeid som ikke er gjort, er avgjørende.

- Den beste arkitekturen, kravene, og designet dukker opp fra selvorganiserte lag med utviklere.
- Med jevne intervaller, skal utviklere reflektere over hvordan bli mer effektiv, deretter koordinere og justere oppførsel/arbeid etter dette. (Avison et al.2006)

Agil /smidig systemutvikling

Systemutviklingstilnærminger kjent som agile, og eller representerer den agile bevegelsen, agile skolen har eksistert for en stund. Det er mange ulike varianter av agil tilnærming men alle disse tilnærmingene tar som utgangspunkt opp de problemene og utilstrekkeligheten ved den tradisjonelle eller fossefallstilnærming til systemutvikling. En av aspektene er relatert til kravspesifikasjoner. (Avison et al.2006)

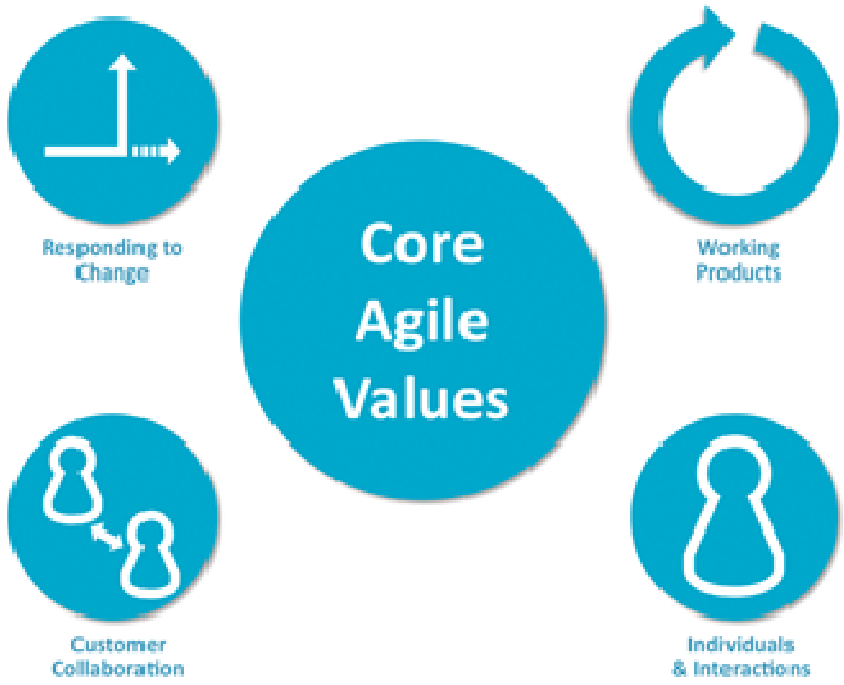
Den agile skolen aksepterer forestillingen om at krav er vanskelig for brukere og ofte kan de ikke artikulere eller definere i detalj deres egne krav til systemet. Brukere vet rett og slett ikke alltid hva de vil ha. Noen ganger vet de hva de vil ha, men når systemet er utviklet for dem, så ser de at det var ikke dette de ville ha. Når bruker har sett systemet, så vet de hva de ikke vil ha og er litt nærmere i sin egen forståelse på hva de egentlig vill ha.

I tradisjonell systemutvikling vil man ha oppdaget dette mot slutten av en lang livssyklus av utvikling, sannsynligvis ved bruker-testing eller ved implementering av selve systemet. Korrigering av applikasjonen på dette tidspunktet er veldig seint og kostbart for den eventuelle bruker eller bedrift. Den agile skolen tror på at krav er så vanskelig å definere at de må bli utviklet på en annen måte. Utviklingen av kravene blir normalt oppnådd ved hjelp av å adoptere en evolusjonær tilnærming, sammen med ”prototype” og en filosofi som omfavner forandring som en norm, og ikke noe som må bekjempes.

Evolusjonær systemutvikling er iscenesatt eller trinnvis tilnærming som levere et system som er komplett i økende grad, dvs. det utvikler seg over tid. Evolusjonær utvikling er noe ganger referert til som inkrementell utvikling. Den første implementering er ikke sett på som det viktigste målet, men bare en del av den pågående utviklingen og forbedringen av systemet til en valgfri løsning på det opprinnelige problemet, eller krav er oppnådd. Etter som systemet modnes vil bare mindre gjentakelser forkomme inntil et skritt av stor endring oppstår som følge av teknologisk utdatering, endring i virksomhet visjon, eller grunnleggende redesigning av nødvendige prosesser. Evolusjonære systemer er ikke i sin endelige form etter den første iterasjonen eller leveringen. Hver leveranse inneholder noe nyttig og brukbart men ikke nødvendigvis komplett.

I den tidlige epoken av programvareutvikling ble det ofte bare bygget et endelig system, fordi å bygge en prototype ble nesten like kostbart og tok nesten like lang til som det endelige systemet og ble derfor ble sjelden gjennomført. Dette har endret seg i takt med den teknologiske utviklingen av programvareverktøy som er tilgjengelig og redusert kostnadene sterkt, samt satt fart i prosessen med å utvikle en prototype. Prototype løser noen av problemene med tra-

disjonell system analyse, særlig klagen ved at brukeren bare ser informasjonssystemet i implementeringsfasen når det er for seint til å gjøre endringer. Hvis en prototype versjon av systemet ikke er utviklet først, da må systemutvikleren eksperimentere med brukeren. Den første versjonen er også den endelige versjonen og dette bringer med seg en åpenbar risiko for svikt, herunder direkte bruker avvisning. (Avison et al.2006)



Bilde 1. Viser kjerneverdier innenfor den agile filosofien.

Agile/smidige metoder

Det er en rekke med metodologier som defineres som agile på en eller annen måte. Disse inkluderer da ”Extreme Programmring (XP), SCRUM, Crystal, Agile Modelling, Adaptive Software Development, Feature Driven Development”. En rekke forfatter av agil tilnærming liker ikke bruken av ordet metodologi for å beskrive dem, men foretrekker å bruke ordet metode eller rammeverk. Grunnen til det er at den agile bevegelsen er sett på som en anti-metodologi generelt sett, altså et tilbakeslag mot metodologier.

Perioden fra midten av 1990 tallet og fremover var preget av en alvorlig revurdering av begreper og praktisering av metoder innen for systemutviklingsfeltet. En del organisasjoner adapterte de ulike metodene, mens andre gav avkall på alt som ble kalt metoder. Metodikk ble

ofte sett på som et universalmiddel for problemet med tradisjonell systemutviklingstilnærminger og ble ofte valgt og vedtatt på feil grunnlag. Noen organisasjoner ønsket bare en bedre prosjektstyringsmekanisme, andre en bedre måte å involvere bruker på, og noen ville gjerne bare ha litt mer disiplin i prosessen. Det var svært usannsynlig at metodologiene noensinne ville oppnå de lovpriste påstandene fra leverandørene og konsulenter. Dette førte til at noen utviklere og organisasjoner avviste metoder i mer generell vurdering og angrep de begrepene de er basert på. (Avison et al2006)

Ekstrem programmering

XP, eller ekstrem programmering er den meste kjente av de agile metodene, den er bygget opp av et sett med enkle og innbyrdes praksiser. XP har også en del verdier og variabler som er fundamentet for denne typen teknikk. Kommunikasjon, de fleste problemer i et prosjekt kan spores tilbake til mangel på kommunikasjon om noe viktig. Enkelhet, hva er den enkleste tilnærmingen`? Tilbakemelding, vil koden bestå testen? Mot, har en mot til å forkaste kode, ikke minste et annet teammedlem sin kode?

De fire variablene er også viktig i et systemutviklings prosjekt. Tid, hvor mye tid er prosjektet tildelt? Omfang, hva er omfanget for systemet som utvikles? Kostnad, hvilke budsjettrammer har prosjektet å forholde seg til? Kvalitet, hvor høye krav stilles til kvalitet?

De tolv praksisene er:

- Kodestandard
- Testing
- Kollektivt kodeeierskap
- Kontinuerlig integrasjon
- Kunden på stedet
- 40-timers uke
- Enkel design
- Par programmering
- Re-faktorering
- Små utgivelser
- Planlegging
- Metafor

En gruppe med ”user stories” erstatter den tradisjonelle kravspesifikasjonstilnærming. Disse blir skrevet av kunde, forholdsvis korte og med et menneskelig eller mindre teknisk språk. Dette utgjør hovedgrunnlaget for planlegging. Hver av bruker historiene (”user stories”) skal ha en test som omhandler aksept. Hvor man tester funksjonaliteten den aktuelle bruker historien utrykker, denne skrives av kunde.

Release planning består av brikker, som er: ”user stories”, spillere som er kunden, utviklere, og trekk som er disse historiene kunde skriver. Utviklerteamet ca beregner hvor lang tid hver ”user story” tar å utvikle, er det mer en 2-4 uker bør den brytes i mindre ”user stories”. Rea-lease plan blir da og estimer hver bruker historie, disse arrangeres i iterasjoner, og de med høy risiko eller høy prioritering legges så tidlig som mulig. Hver utgivelse bør være så liten som mulig å inneholde de mest verdifulle ”user stories”. Iterasjonsplanlegging består av å bryte ned hver ”user story” i utviklingsoppgaver, for å finne ut konkret hva som må gjøres for å utvikle en slik funksjon til systemet. Hver av disse utviklingsoppgavene må være små nok til at alle utviklere i laget forstår hva denne innebærer og man er enig. Hver iterasjon må ende i et fungerende system.

Informasjonsmøtene utføres ved at alle står, det er for å få møte til å gå fort og effektivt. Alle utviklere deler informasjon med hverandre, om hva de har gjort, hva de skal gjøre og hvilke problemer de har møtt på og endringer til disse.

En form for kladdeløsning brukes også ofte kalt ”spikes”, som er en hjelpemetode til å finne løsninger på tunge tekniske eller designmessige problemer. Dette er vanligvis et enkelt pro-gram for og utforske mulige løsninger på et problem. Dette brukes til å prøve ut ideer, løs-ninger og komme nærmere målet, samt etablere forståelse av domenet.

Par programmering, to programmerere per stasjon, dette reduserer potensielle risikoer når tekniske vanskeligheter truer systemet. En vil da programmere og den andre vil tenke mer strategi som heller mot retningen hvor en reflekterer over om tilnærmingen vil fungere. Disse rollene bør byttes ofte, et annet prinsipp som involverer å flytte personer rundt. Grunnen til å flytte personer rundt er fordi kunnskapen blir spredt mellom alle utviklere og ikke minst kunnskapen om systemet i sin helhet. (Avison et al2006)

Scrum

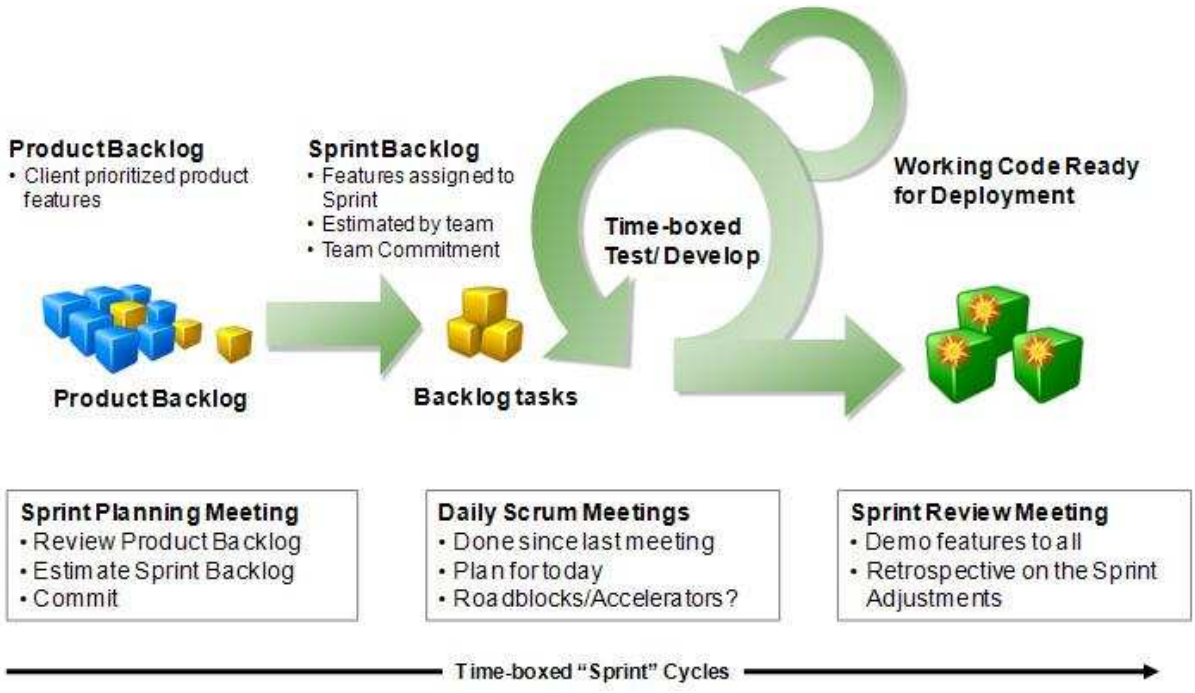
Scrum er oppkalt etter ”scrum” i rugby som er en mekanisme for å starte spillet på nytt etter et tilfeldig brudd, som når eventuelt ballen går utenfor banen. Scrum er et prosjekt rammeverk for styring av agile metoder/prosjekter. Det viktigste målet er å levere programvare av høy kvalitet etter hver endte iterasjon. Den er basert på ”sprinter” som er i mellom 2 til 4 uker. En grunnleggende faktor er at prosjektlagene er selvorganisert, det vil si at man organiserer seg i utgangpunktet basert på målet for ”sprinten”, og seinere daglig igjennom scrum-møter. Møte-ne blir arrangert daglig, de er korte og alle står i 10-15 min. Hvert medlem må svare på tre spørsmål, disse vil ta opp enkle ting som, hva har blitt gjort? Hva er planen videre? Er det noen veisperringer? Denne rapporteringen er ikke et statusmøte, mer et møte om hva man har gjort, hva man vil gjøre og hvilke hindringer en har møtt på. Når hindringer er rapportert be-stemmer teamet seg for hvordan det skal håndteres. Når flere lag er involvert vil det forkom-me hierark av møter. Hvis det er 3 lag involvert i arbeidet på tre separerte, men relaterte ”sprints” så vil hvert team holde deres eget møte. Et av medlemmene fra hvert av lagene vil så ha et annet møte for å koordinere seg sammen med de andre lagene.

Det eksisterer også en del roller innefor en scrum praksis, produkt eier, scrum-master og team medlemmer. Produkt-eier er ansvarlig for å representere kunden eller bruker av programvaren utviklerteamet bygger. Ideelt sett er dette den faktiske kunden, når kunden ikke er tilgjengelig vil et bindeledd som representerer kundens interesse bistå i kundens sted. En scrum-master sin rolle er primært å lette, rapportere og få teamet til å fokusere på de oppgavene som er høyest prioritert og ikke minst å få vekk veisperringer for systemets fremdrift. Team medlemmer inkluderer utviklere, arkitekter, prosjektledere, testere, database administratorer m.m.

Produkt backlog(bilde 2) er en liste over alle kravene for å levere et system, disse er vanligvis definert på et høyt nivå og inkluderer estimater brukt til omgang av ”sprint” ordrereserven.

Ved starten av hver Sprint, bryter prosjektgruppen ned elementer fra Release Backlog, starter på toppen (de viktigste), og legger disse inn i Sprint Backlog. Når nok elementer har blitt valgt til å fylle i ”sprinten”, så blir Backlog låst. Estimatene inkluderer total tid til å fullføre hvert element, inkludert analyse, design, koding, testing, dokumentasjon, m.m. Det er et annet element relatert til ordrereserven som er nøkkelen til Scrum.” Burndown Chart” brukes for å indikere antall gjenstående Sprint Backlog elementer. En daglig post som vedlikeholdes med hensyn til lagenes fremgang.

På slutten av en Sprint, møter laget med alle interessenter til å vise hva arbeidet som er ferdig, og å vurdere prioriteringer for neste Sprint. Siden Scrum fokuserer på prosjektledelse ved et prosjekt og spesifiserer ingen teknisk praksis, integrerer det godt med andre Agile metoder. Det er ofte kombinert med XP, men vil fungere fint med andre tilnærminger også.(Coffin et al.2010)



Bilde 2. Viser Backlogen sin syklus i henhold til scrum rammeverket.

Kritikk på agile metoder

En del av kritikken mot agile metoder heller mot mangelfull vektlegging på design og dokumentasjon. Dokumentasjon er alltid viktig, det er det skrevete bevis på hva som er gjort og hvordan det ble gjort. De agile metodene legger mer vekt på fungerende programvare og en enkel dokumentasjon, i stedet for et dokument som beskriver alt. Et annet aspekt er hvis kundene eller den som representerer kunden sine interesser ikke vet hva de vil ha til resultat, dette kan ha grunnlag i mangelfull innsikt i bedriften og eller mangelfull kunnskap om hvilke funksjoner en ønsker.(Sousa 2009)

På grunn av den integrerende tilnærmingen er det vanskelig for noen å finne ut nøyaktig hvor prosjektet står. I et typisk miljø vil som oftest en leder vite når en fase er ferdig, på grunn av ulike iterasjons trinn kan det være vanskelig å forstå om prosjektet er i rute. Agil ledelsesmetode passer ikke til store lag, men bra til mindre eller mellomstore grupper. Agil utvikling krever dyktige og høyt motiverte individer, som kanskje ikke alltid eksisterer.(Sanja 2005)

” The fact is, agile by itself is just one tool in the toolbox that should be applied with other implements of the trade. In my experience, the problem comes in most often because small- and mid-sized organizations experience brilliant success with agile and then assume it can work everywhere “(Beckman, 2010). Å endre krav fører med seg en tung byrde, særlig når det gjelder på tvers av den organisatoriske effekten til markedsføring, budsjett, kvalitetsstyring og kunde. Endringer eller kutt vil også endre kontroll, krav til ledelse og konfigurasjonsstyring fra prosessen. Disse endringene kan føre til langsiktige katastrofer som de agile metodene kan overse på kortsikt. De temaene som omhandler å endre/redusere struktur og kontroll vil kutte ut mange av de tradisjonelle prinsippene. Faren manifesteres ofte etter hvert som småskala agile prosjekter er vellykket og man skal innføre de agile metodene over på store eller større prosjekter. Da vil viktige komponenter som kvalitetssikring forsvinne, per i dag er det ingen agile metoder som integrerer omfattende kvalitetssikring, Software testing er ikke kvalitetssikring, Software testing som ofte blir en ettertanke, og risikostyrings-programmer har en tendens til å bli anset som ”fuzzy disipliner”, likevel er det disse prosessene som klarer å sette mennesket på månen eller sørger for at kjernekraftverk systemer ikke mislykkes etter levering. (Beckman, 2010)

Oppsummering og Konklusjon

Fossefall modellen er nok den første modellen man hører om innen systemutvikling generelt, denne modellen var en av milepælene. Modellen har fått masse kritikk og en gruppe eksperter har også lage et mot argument til den og tidsepoken etter den. Denne modellen var en av de første store utviklingsmodellene informasjonsteknologien så. Og hvis man er først på banen så er det lettere for andre å finne ulike tilnærminger som kan virker bedre på sin måte og ta ulike deler til seg. Informasjonsteknologien har gått som en rakket siden 1950 årene, og da er det naturlig at ulike modeller, teknikker og innovasjoner trer også frem i takt med farten.

Den agile alliansen og den smidige systemutviklingen viker fra den tradisjonelle modellen, og omfavner endringer, kommunikasjon og hyppige levering. Integrering av bruker og kollektivt kodeeierskap er ”must” for smidig utvikling mot suksess.

SearchSoftwareQuality.com sin `2008 Agile trender undersøkelsen, fant ut at mange mennesker fremdeles følger de tradisjonelle metodene og teknikkene. Når spørsmålet om hvilken utviklingsprosess de ansatte, og bedriften valgte var det ganske jevnt mellom de agile og de tradisjonelle. 46% svarte agile, mot 44% som svarte for fossefalls modellen.(Davidson 2008)

Det eksisterer et stort antall av metoder for å konstruere informasjonssystemer, dette vil da si at det også eksisterer et stort antall av ”beste” måter for å utvikle systemer på. De tradisjonelle metodene passer bra når alle stegene blir nøye gjennomført og ingen form for anger, eller endring må skje, de agile metodene passer fint til systemer hvor endring, eller nye krav kan dukke opp. Selv om disse modellen har mye like faser i seg, blir de fulgt opp og gjennomført ulikt. Noe som er verd å merke seg er at ytre faktorer alltid vil være tilstede i alle disse modellene/teknikkene for å utvikle et system. De ytre faktorene vil være kunder, å følge en tidsplan, økonomien som er investert, oppdatering av teknologi, kompetansen på laget, og andre faktorer som dukker opp underveis. Veien til suksess innenfor systemutvikling har mange gater og kryss før systemet kan overleveres.

Kildehenvisning

Avison, D, Fitzgerald, G (2006) “ Information systems development, methodologies, techniques & tools ” 4th edition.

Beck,K. Beedle, M. Bennekum,A,V. Cockburn, A. Cunningham, W.

Fowler, M. Grenning, J. Highsmith, J. Hunt, A. Jeffries, R. Kern, J.

Marick, B. Martin, R, C. Mellor, S. Schwaber, K. Sutherland, J.

Thomas,D . (2001) “Manifesto for agile software development”

Tilgjengelig fra: <http://agilemanifesto.org/>

[lastet ned 12 .oktober 2010].

Beckman, Z. (2010) ” Why Agile isn’t enough (and why it doesn’t work”

Tilgjengelig fra:< http://www.rational-scrum.com/2010/04/why-agile-isnt-enough-and-why-it-doesnt-work/>

[Lastet ned 15.november 2010].

Bilde 1:

Tilgjengelig fra: < http://www.getzcope.com/blog/2009/10/28/core-agile-values/>

[lastet ned 30.oktober 2010].

Bilde 2: Tilgjengelig fra: < http://agilescrum.biz/agile-scrum/agile-scrum-1.jpg>

[Lastet ned 18. okober 2010].

Boddy, D. Boonstra, A. Kennedy, G. (2008)

"Managing Information Systems Strategy and organization" third edition .

Coffin, R. Lane, D.(2010)" A Practical Guide to Seven Agile Methodologies"

Tilgjengelig fra: <http://www.devx.com/architect/Article/32761/0/page/3>

[Lastet ned 15. november 2010].

Davidson, M.(2008) Survey: "Agile interest high, but waterfall still used by many"

Tilgjengelig fra:
<http://searchsoftwarequality.techtarget.com/news/article/0,289142,sid92_gci1318992_mem1,00.html>

[Lastet ned 6.september 2010].

Martin, R, C. Newkirk, J, W. Koss, R, S. (2003).

"Agile software development, Principles, patters, and practices" .

Sanja, A, V. (2005) "Overview of Agile Management & Development "

Tilgjengelig fra: <http://www.projectperfect.com.au/info_agile_programming.php>

[Lastet ned 3. november 2010].

Sousa, S, D.(2009) " The Advantages and Disadvantages of Agile Software Development"

Tilgjengelig fra: <http://www.my-project-management-expert.com/the-advantages-and-disadvantages-of-agile-software-development.html>

[Lastet ned 10.november 2010]

Weaver, P. (2004) "Success in your project, a guide to student development projects".