Programvareutviklingsprosessen -
organisering, styring, kontroll

OMNI/02/00

Egil P.Andersen, NR

Oslo
August 2000

Norsk Regnesentral
ANVENDT DATAFORSKNING

NR-notat/NR Note

Tittel/Title: Programvareutviklingsprosessen - organisering, styring, kontroll

Dato/Date: 31.august
År/Year: 2000
Notat nr: OMNI/02/00
Note no:

Forfatter/Author: Egil P.Andersen (NR)

Sammendrag/Abstract:

Dette notatet gir et kortfattet resyme av boken *'Managing the Software Process'* av Watts S.Humphrey (Addison-Wesley, 1989(reprint 1990)). Boken beskriver en metode for å vurdere på hvilket modenhets- elle kvalitetsnivå en programvareutviklingsorganisasjon befinner seg m.h.p. programvareutviklingsprosessen. Den er grunnlaget for en metode, i ettertid kalt *CMM (Capability Maturity Model)*, utviklet av Software Engineering Institute(SEI). SEI tilhører Carnegie Mellon University, og er en organisasjon opprettet og finansiert av det amerikanske forsvaret med det mål å få bedre kontroll med programvareutviklingsprosjekter

Litteratur:

Det er mye litteratur referert i 'Managing the Software Process', men den er fra 1989 og derfor antagelig noe utdatert.

Software Engineering Institute: *http://www.sei.cmu.edu/FrontDoor.html*

Emneord/Keywords: Programvareutvikling, QA, utviklingsprosesser

Tilgjengelighet/Availability: Åpen

Prosjektnr./Project no.: -

Satsningsfelt/Research field: Programvareutvikling

Antall sider/No. of pages: 31

Norsk Regnesentral / Norwegian Computing Center
Gaustadalléen 23, Postboks 114 Blindern, 0314 Oslo, Norway
Telefon 22 85 25 00, telefax 22 69 76 60

Contents

1. Et Rammeverk for Modenhetsvurdering 3

1.1 Modenhetsnivåer for Programvareutviklingsprosessen 3
1.2 Forbedring av Programvareutviklingsprosessen 4

2. Prinsipper for Prosessforbedring 5

2.1 Menneske versus Prosess 5
2.2 Grunnleggende prinsipper for prosessforbedring 5
2.3 Vanlige misforståelser om programvareutvikling 6
2.4 Strategi for forbedring 7

3. Vurdering av Utviklingsprosessen 8

3.1 Oversikt 8
3.2 Prinsipper for Prosessvurdering 8
3.3 Utførelse av Vurderingsprosessen 8

4. Den Initsielle Prosessen 10

5. Ledelse av Programvareutviklingsorganisasjoner 11

6. Prosjektplanen 12

6.1 Prosjektplanlegging 12
6.2 Måltall for Størrelse 12
6.3 Estimering 12
6.4 Prosjektkontroll 13
6.5 Utviklingsplan 13

7. Konfigurasjonsstyring - Del 1 14

8. The Software Quality Assurance Group (SQA) 15

9. Standarder og Prosedyrer 16

9.1 Hensikten med Standarder 16
9.2 Etablering av Stadarder 16

10. Formelle Inspeksjoner og Gjennomganger 17

10.1 Ulike typer inspeksjoner 17
10.2 Gjennomføring av inspeksjoner 17

11. Testing av Programvare 19

11.1 Prinsipper for Testing 19
11.2 Testplanlegging 20

12. Konfigurasjonsstyring - Del 2 21

13. Modeller av Utviklingsprosessen 22

14. The Software Engineering Process Group (SEPG) 23

15. Datainnsamling og Analyse 24

15.1 Prinsipper for Datainnsamling 24
15.2 Måltall for Programvare 25
15.3 Dataanalyse 25

16. Kvalitetssikring og Kvalitetskontroll 26

16.1 MÅLTALL FOR KVALITET 26
16.2 ETABLERING AV KVALITETSSIKRING 27
17. Å Forebygge Feil 28
18. Prosessautomatisering 29
19. Kontrakter om Programvareutvikling 30
19.1 TILLIT 30
19.2 ENDRINGER MÅ BEHANDLES SOM ET NATURLIG ELEMENT I UTVIKLINGSPROSESSEN 30
20. Oppsummering 31
20.1 LOKALE TILPASNINGER 31
20.2 KONFLIKTSKAPENDE? 31
20.3 KONSEKVENSER AV MISLYKKET PROSESSFORBEDRING 31

Del 1 : Modenhetsnivåer i Programvareutviklingsprosessen

1. Et Rammeverk for Modenhetsvurdering

1.1 Modenhetsnivåer for Programvareutviklingsprosessen

'Managing the Software Process' definerer følgende 5 nivåer for modenhet m.h.p programvareutviklings-prosessen, hvor målsetningen er å oppnå en optimaliserbar prosess (nivå 5).

Den Initsielle Prosessen (Nivå 1)

Karakteristika: Uforutsigbare resultater m.h.p kostnader, tidsplaner og kvalitet i produkt. I beste fall tilfeldig, i verste fall kaotisk.

Det er her gjerne få eller ingen formaliserte rutiner, estimat eller planer. Selv om planer finnes vil det ikke være noe system for å følge dem opp. Ved kriser vil man typisk falle tilbake til 'koding og testing' og droppe bruk av eventuelle teknikker og metoder som ellers skal benyttes.

Forbedringstiltak: Planlegging, endringskontroll og konfigurasjonsstyring.

Den Repeterbare Prosessen (Nivå 2)

Karakteristika: Intuitiv - kostnad og kvalitet høyst variabel. Noenlunde kontroll med tidsplaner og avtaler. Uformelle og ad-hoc prosessmetoder og -prosedyrer.

Fordeler oppnådd relativt til nivå 1 er først og fremst basert på tidligere erfaring med tilsvarende arbeid. Gjennomføring av nye prosjekter med f.eks ny teknologi vil derfor representere de samme risikofaktorer som for nivå 1.

Forbedringstiltak: Innføring av standarder, formelle inspeksjoner og tester, utvidet konfigurasjonsstyring, formalisering av utviklingsprosessen.

Den Definerte Prosessen (Nivå 3)

Karakteristika: Kvalitativ - forutsigbare kostnader og tidsplaner. Forbedret men uforutsigbar m.h.p kvalitet.

Det er nå grunnlag for å kunne studere prosessen i detalj og finne ut hvordan den kan forbedres. Foreløpig lite data om eksakt status for prosjekter, eller hvor effektiv prosessen egentlig er.

Forbedringstiltak: Innføre kvantitative kvalitetsmål, utføre målinger, foreta kvantitative analyser.

Den Kontrollerte Prosessen (Nivå 4)

Karakteristika: Kvantitativ - noenlunde forutsigbar også m.h.p kvalitet.

Status for prosjekter, effektivitet av prosessen, osv, er kvantifiserbart. Det er grunn til å forvente store kvalitetsforbedringer ved overgang fra nivå 3.

Forbedringstiltak: Evne å utnytte kvantitative mål til systematisk prosessforbedring.

Den Optimaliserbare Prosessen *(Nivå 5)*

Karakteristika: Kvantitativ basis for kontinuerlig forbedring.

Overgang fra nivå 4 til nivå 5 er et paradigmeskifte. Opptil nivå 4 er det mest fokus på målinger av produkter og prosjekter for forbedring av disse. Nå vil data finnes tilgjengelig for tuning av selve prosessen og forbedringer av denne. Effekten av å applisere prosessen på et gitt produkt kan måles og benyttes for videre prosessforbedring.

Forbedringstiltak: Sikre vedlikehold av prosessen over tid og gjennom endringer i prosess og prosjekter.

Kommentarer

Denne inndelingen i nivåer og aktiviteter er valgt basert på det erfaringsgrunnlag som finnes. Flere av de aktiviteter som beskrives på høyere nivåer vil være aktuelle fra start, men denne inndelingen er ment å beskrive hvilke prioriteringer som bør gjøres, og hvilke områder det er fornuftig å fokusere mest på ved overgang til et høyere nivå, og hvor man kan ha størst forventning om forbedring. Prioritering er essensielt for å konsentrere innsatsen og derved kunne oppnå resultater.

1.2 Forbedring av Programvareutviklingsprosessen

Ifølge en artikkel (1) fra 1994 er fordelingen mellom de ulike nivåer som følger (utplukket er imidlertid ikke spesifisert):

Nivå 1 - 75%
Nivå 2 - 15%
Nivå 3 - 8%
Nivå 4 - 1.5%
Nivå 5 - 0.5%

Ifølge en spørreundersøkelse (2) i 1996 blant 166 norske programvareutviklingsbedrifter gjør ca.25% ingen forsøk på å forbedre sin nåværende prosess. Ca.50% benytter ulike metoder for kvalitetssikring som aktivitetsplaner, prosjektkontroll og -evaluering, inspeksjoner, rutiner for testing, osv. Mellom 25-30% foretar målinger basert på tester. 17% benytter ISO/TickIT, mens bare 8% benytter metoder for prosessvurdering og -forbedring. Av disse benytter 4 bedrifter CMM, mens 2 benytter Bootstrap.
Videre viser undersøkelsen at det stort sett er store, veletablerte bedrifter som bruker slike metoder, mens flest små, yngre bedrifter ikke benytter dem.

Det viktigste trinn er kanskje 1->2 da dersom ser gevinst her vil det gi motivasjon for videre forbedring.
Overgang fra nivå 1 til 2, og fra nivå 2 til 3, kan forventes å ta ca.1-3 år selv med de beste forutsetninger tilstede.

(1) Capers Jones
Assessing the Software Engineering Institute: Gaps in the SEI Programs
Software Productivity Research, Burlington, MA, USA, 1994

(2) Karl Heinz Kautz, Even Åby-Larsen
Diffusion and Use of Quality Assurance and Software Process Improvement Approaches in Norway: A Survey-based Study
Rapport Norsk Regnesentral, 1996

2. Prinsipper for Prosessforbedring

Følgende er nødvendig for en forbedring av programvareutviklingsprosessen:

1) Evaluering av nåværende status ('hvor er vi?')
2) En visjon av den ønskede prosess - en ide om målsetningen ('hvor skal vi?')
3) En liste, etter prioritet, over tiltak påkrevd for forbedring av prosessen ('hvordan skal vi komme dit?')
4) En plan for gjennomføring av de aktuelle tiltak ('husk kart')
5) Tilstrekkelig ressurser til å kunne gjennomføre planen
6) Goto 1) ('vi skal videre...')

2.1 Menneske versus Prosess

'Managing the Software Process' fokuserer primært på tekniske og ledelsesmessige problemstillinger og metoder for prosessforbedring. Selv om det ikke foretas noen dypere vurderinger av menneskets plass i en slik prosess er dette hele tiden kommentert.

Programvarekvalitet er usedvanlig mye avhengig av de som faktisk bygger produktet. I så måte mye likt det f.eks å bygge et hus versus masseproduserte fabrikkvarer.

En spesielt viktig faktor er at det nesten alltid er slik at desto bedre utviklerne kjenner applikasjonsdomenet, desto bedre blir kvaliteten og sannsynligheten for suksess. Verdien av dette kan ikke overvurderes.

Bør imidlertid ikke fokusere alt for mye på talent alene. Det er alltid for få av de dyktigste, du har antagelig de beste du kan få for øyeblikket, og med god ledelse kan man få de fleste til å gjøre en enda bedre jobb. Prosessforbedring kan også gi bedre utnyttelse av personlig talent. Ledelsen forstår bedre hvor støtte og hjelp er påkrevd. Det muliggjør kommunikasjon i presise og kvantitative termer. Det finnes et rammeverk for forståelse av prosessen, de resultater som oppnås, og ikke minst hvordan denne kan kontinuerlig forbedres.

Prosessforbedring kan gjøre bedriften attraktiv for dyktige personer dersom blir kjent for kvalitet. Kan derfor være en selvforsterkende prosess i så måte.

Det er viktig å skjelne mellom disiplin og regimentering. Disiplin gir utvidet rom og mulighet for kreativitet ved at unngår unødvendige problemer og kriser, mens regimentering kveler kreativitet. Prosessdisiplin skal gi kontroll med settingen hvor arbeidet skal utføres, men ikke utidig detaljkontroll med hvordan arbeidet skal utføres. Pr.idag synes behovet for mer disiplin i programvareutvikling å være viktigere enn farene for regimentering; dvs muligheten for å frigjøre tid til mer kreativitet synes å måtte veie mer enn faren for endringer som kan begrense kreativiteten.

Utgangspunktet for en prosessforbedring skal alltid være at det er prosessen og ikke de ansatte som er den underliggende årsak til feil og problemer, og som derfor skal forbedres. Kan anta at de fleste har et sterkt underliggende ønske om å gjøre en god jobb dersom de gis muligheten til dette.

2.2 Grunnleggende prinsipper for prosessforbedring

Følgende kan sies å være de grunnleggende prinsipper for suksessfull endring av programvareutviklings-prosessen:

- Større endringer må ha full støtte fra toppledelsen.
  Dette er nødvendig for å kunne starte prosessen, og være sikret tilstrekkelig ressurser og prioritet.

  Prosessproblemer er primært et ledelsesansvar og -problem. Ledelsen må foreta prioriteringene, tilby ressursene, og alltid gi prosessen full støtte. Til gjengjeld må de insistere på resultater.

- Alle må inkluderes i prosessen - lagånd er viktig.

  Menneskene involvert er den viktigste ressursen. Deres ønske om å gjøre godt arbeid må tas hensyn til, og målet er å forbedre prosessen, ikke menneskene involvert.

- Effektive endringer krever klare målsetninger og kunnskap om den nåværende prosessen ('du må vite hvor du er før du kan bruke kartet').

En vurdering av nåværende prosess er en læringsprosess hvor det viktigste er å kunne foreta en fornuftig prioritering.

Særlig i organisasjoner på lavere nivåer er ikke tekniske implementasjonsmessige problemer de viktigste. Dette ikke fordi de ikke er viktige, men det er andre og større problemer som er viktigere. Organisasjons-messige og ledelsesmessige problemer som er ofte mest fremtredende og må derfor prioriteres. Dette gjelder endringskontroll, planlegging, system design og kravspesifikasjoner.

- Menneske-intensive prosesser er aldri statiske. Endring er en kontinuerlig prosess og krever periodevise justeringer.

  Justeringer for forbedring bør imidlertid aldri foretas under press og i kriser.

- Kvaliteten på utviklingsprosessen degraderer over tid uten målrettet innsats.

  Fysikkens lov om økning i entropi synes også å gjelde for utviklingsprosesser. Uten varig målrettet innsats over tid vil ikke forbedringsgevinsten bestå.

- Prosessendringer og forbedringer må sees på som en langsiktig investering ('utgift til inntekts ervervelse').

2.3 Vanlige misforståelser om programvareutvikling

Følgende er noen vanlige misforståelser om programvareutvikling:

- Vi må starte med en presis og komplett kravspesifikasjon.

  Det er feilaktig å tro at kravspesifikasjonen kun er kundenes ansvar, og at utvikling ikke kan starte før kravspesifikasjonen er 100% komplett.

  Det er absolutt et unntak dersom en kravspesifikasjon forblir uendret under utviklingsprosessen. Produktet påvirker sin omgivelse, noe som medfører endringer, som igjen endrer produktet, som igjen endrer omgivelsen, osv.

  Det må derfor alltid tas høyde for endringer, og rutiner og metoder for håndtering av dette må være klarlagt i utgangspunktet fremfor å sees på som uforutsette problemer når de (alltid) dukker opp.

- Dersom produktet tilfredsstiller testene må det være iorden.

  Ettersom det lages stadig større og mer komplekse programvareprodukter vil testing bli en stadig dårligere indikator på systemkvalitet fordi det vil kunne dekke stadig færre aspekter av det som oppfattes som kvalitet ved et produkt.

- Programvarekvalitet kan ikke måles.

  Det finnes ingen måltall som representerer programvarekvalitet fullt ut, men det finnes heller ikke for de fleste andre produkter som produseres. Vi må derfor søke etter kombinasjoner av måltall som sammen gir en god indikasjon på kvalitet.

- Problemene skyldes tekniske problemer.

  Erfaring viser at problemene som det har høyest prioritet å løse sjelden er tekniske problemer.

- Vi trenger ansatte som er dyktigere i jobben sin.

  Utgangspunktet for en prosessforbedring skal alltid være at det er prosessen og ikke de ansatte som er den underliggende årsak til feil og problemer, og som derfor skal forbedres. Kan anta at de fleste har et sterkt underliggende ønske om å gjøre en god jobb dersom de gis muligheten til dette.

- Ledelse av programvareutviklingsprosjekter er noe helt nytt.

  Programvare betraktes av mange ledere som noe nesten mystisk og utilgjengelig. Ledelse av programvare-prosjekter bør imidlertid være helt tradisjonell; med detaljerte planer, oppfølging, evaluering, osv.

2.4 Strategi for forbedring

En forbedringsprosess må starte med innsats for å forstå nåværende status, dvs årsaker til problemer og potensial for forbedring. Dette kan gjøres ved en vurderingsmetode som beskrevet i kapittel 3.

Det finnes ulike roller for mennesker involvert i en endringsprosess. 'Champions' er de som initsierer endringsprosessen. De gjør ledelsen oppmerksom på behovet og sikrer støtte til arbeidet. Dette kan være en meget tung prosess, men de fleste suksessfulle endringsprosesser har hatt noen 'champions' som har kjempet prosjektet gjennom.

Den øverste ledelsen spille rollen som 'sponsorer', og dette er en meget viktig rolle. Noen med tilstrekkelig autoritet er nødvendig for å sikre gjennomføring ved tildeling av ressurser og utvetydig støtte underveis.

Neste trinn er å finne 'agenter' som skal lede planleggingen og organisere gjennomføringen av endringsprosessen. De styrer ressursene, fordeler arbeidsoppgaver, og søker støtte hos 'sponsorene' når det er nødvendig. Kapittel 14, om Software Engineering Process Group, beskriver ansvar og oppgaver for 'agentene' under endringsprosessen.

De sentrale elementer i en effektiv endringsprosess er:

- Detaljerte og realistiske planer
- Implementere endringene i et passe tempo; ikke så langsomt at stopper opp, men ikke så hurtig at ødelegger den stabilitet som er nødvendig for gjennomføring av prosjekter.
- Kommunisere planer, status, resultater, etc til organisasjonen.

Følgende illustrerer en slags generisk oversikt over vanlige elementer som går igjen i de fleste aktiviter for prosessforbedring beskrevet i boken (de ulike aktiviteter normalt har et undermengde av disse elementene):

- Målsetning (basert på problemer identifisert)
- Sikre støtte hos ledelse
- Definer prioriteringer
- Planlegging (standarder, prosedyrer, måltall, osv)
- Alloker ressurser og ansvar
- Opplæring og kursing
- Start implementasjon
- Review/inspeksjoner, kontroll med status og fremdrift, godkjenning
- Dokumenter og rapporter
- Foreta periodiske justeringer etter behov - alltid endringer

3. Vurdering av Utviklingsprosessen

3.1 Oversikt

Hensikten med å foreta en prosessvurdering ('software process assessment') er at organisasjonen skal lære om seg selv og forbedre seg selv ved å identifisere sine mest kritiske problemer. Derved kan det settes opp en liste av forbedringstiltak angitt etter prioritet.

Det særlig viktig å få med de ansatte som setter dagsorden i organisasjonen i de ulike avdelinger og organisasjonsnivåer for å sikre at målsetningen godkjennes og respekteres av alle ansatte.

Vurderingen foretas som en serie intervjuer for å lære om ansattes problemer, deres vurderinger og å få frem kreative ideer til løsninger. Den bør utføres av noen som har trening og erfaring med slik vurdering, fortrinnsvis med en blanding av lokalt ansatte og utenforstående, men ingen ansatte bør være med å vurdere egne prosjekter.

Målet er å identifisere hvilke områder som det er viktigst å gjøre noe med, og å komme med retningslinjer og råd for hvordan forbedring kan utføres.

Vurderingen utføres i 3 faser:
- Forberedelse
- Gjennomføring
- Rådgiving

Under forberelsene må øverste ledelse forplikte seg til å delta personlig, og til å enten gjennomføre de råd som gis eller alternativt gi en begrunnelse for hvorfor ikke.

Gjennomføringen kan ta fra noen dager opptil 1-2 uker.

I siste fase gis det råd til tiltak og deretter opprettes en gruppe med ansvar for planlegging og gjennomføring av de aktuelle tiltak.

3.2 Prinsipper for Prosessvurdering

Følgende er noen grunnleggende prinsipper for vurdering av utviklingsprosessen:
- Vurderingen må baseres på en prosessmodell.
  Det må finnes en standard å sammenligne organisasjonens utviklingsprosess med (et kart å gå etter). De 5 modenhetsnivåene kan utgjøre et slikt rammeverk.

- Konfidensialitet er avgjørende.
  I den grad mulig bør ikke enkelt-ansatte eller enkelt-prosjekter identifiseres, og synspunkter som kommer frem skal ikke rapporteres videre.

- Øverste ledelse må være personlig involvert da de er ansvarlig for organisasjonens overordnede prioritering.

- Alle ansatte må behandles med respekt.
  En liten gruppe kan ikke selv løse en organisasjons problemer på noen få dager eller uker. De skal derimot fungere som en katalysator for selv-forbedring basert på at de ansatte er kloke, motiverte og kreative, og at målet er å systematisere den kunnskap som allerede finnes i organisasjonen.

- Vurderingene må være handlings-orientert og rettet mot konkrete forbedringstiltak.

3.3 Utførelse av Vurderingsprosessen

Vurderingen må fokusere på hvordan ting faktisk gjøres, hvilke problemer som møtes, og hvilke resultater som oppnås, ikke på hvilke metoder o.l som idag finnes for hvordan ting egentlig skulle vært gjort.

Det er viktig med åpne spørsmål som 'Dersom du kunne endre 1 aspekt av den nåværende prosessen, hva ville du da ha endret, og hvorfor?'.

Svar må etterprøves, men ikke på en slik måte at ansatte føler at de blir mistrodd e.l.

Resultatene av vurderingen skal presenteres skriftlig, og det bør ikke være mer enn rundt 10 tiltak som utdypes, og kanskje 3-4 som høyprioritetstiltak.

Basert på resultatdokumentet må det utarbeides handlingsplaner, ansvar for gjennomføring må tildeles, og noen i ledelsen må gis ansvar for oppfølging og gjennomgang av status og fremdrift.

Normalt bør organisasjoner foreta en ny vurderingsrunde 1-2 år etter de initsielle handlingsplaner er laget og godkjent. Dette for å vurdere fremdrift og for å sette nye mål og prioriteringer.

Følgende er noen vanlige årsaker til at endringstiltak ikke settes ut i drift eller ikke har noen effekt:
- Utilstrekkelig støtte fra ledelse
- Konflikt med andre prioriterte mål (tids- og kostnadsplaner)
- Manglende oppfølging

De viktigste kriterier for sannsynlig suksess er:
- En 'aggressiv' leder til å lede arbeidet
- De som utfører arbeidet har kunnskaper om prosessforbedring
- En klar og utvetydig målsetning

4. Den Initsielle Prosessen

Det finnes mange graderinger av den initsielle prosessen, dvs nivå 1, men de har alle det til felles at de mangler en planlagt, veldefinert, kontrollert og disiplinert utviklingsprosess.

Det hender at levering foretas i tide, innen kostnadsrammene og med akseptabel kvalitet, men dette skyldes da gjerne 'heroisk' innsats av enkeltindivider fremfor organisasjonen som sådan.

Prosessen er karakterisert ved mange kriser like før levering, repetisjon av mer eller mindre de samme problemer, utilstrekkelige ressurser, dårlig koordinering og vage statusrapporter.

En manglende avtale-disiplin er den vanligste, og kanskje vanskeligste, årsak til kaotisk oppførsel. Det gjøres avtaler om levering innen visse tids- og kostnadsrammer uten at det legges en detaljert plan for hvordan dette skal realiseres. Underveis dukker det da gjerne opp ubehagelige overraskelser. Det gjøres avtaler som ikke kommuniseres slik at de som skal gjøre arbeidet ikke får informasjon om dette før det er for sent. Problemene blir gjerne selvforsterkende - for å ta inn det tapte loves enda mer innen enda strammere rammer, osv (som gambling med kvitt-eller-dobbelt).

'Guruer' som styrer det meste fra eget hode er også et vanlig problem. Både grunnet mye avhengighet av enkeltpersoner, og at fremgangsmåten skalerer meget dårlig.

Etter å ha gjennomført et prosjekt med stor suksess tror man at man kan klare alt. Man glemmer lett at det oftest er fundamentalt forskjellige problemer involvert om man går fra et mindre prosjekt til et større selv om det er innen samme applikasjonsdomene.

Løsningen på de umiddelbare problemer kan skisseres som følger:

- Anvend systematisk prosjektledelse. Arbeidet må estimeres, planlegges og ledes.
- Innfør formell endringskontroll av kravspesifikasjon, design, implementasjon og testing.
- Det er nødvendig å forsikre seg om at essensielle prosjektaktiviteter utføres slik de skal.

Dette er normalt de prioriterte tiltaksområdene for å komme fra nivå 1 til nivå 2. Det sentrale her er innføring av planlegging som et viktig element i utviklingsprosessen. Uten planlegging blir man lett offer for 'utviklingskrefer' man ikke klarer å forutse på forhånd.

Enhver programvareutviklingsorganisasjon trenger også standarder, produkt gjennomganger og inspeksjoner, testing, osv, men dette har lavere prioritet før man når et høyere nivå hvor prosjekt planlegging og ledelse er forbedret.

Del 2 : Den Repeterbare Prosessen

5. Ledelse av Programvareutviklingsorganisasjoner

De grunnleggende oppgavene for ledelse av programvareutviklingsorganisajoner er:

- Etablere en avtaledisiplin.
- Det må eksistere et system for løsning av naturlige konflikter mellom prosjekter og mellom prosjekter og den øvrige organisasjon.
- Det må eksistere et system for oversikt, kontroll og gjennomgang av prosjekter m.h.p status og fremdrift.

En avtale er typisk en enighet mellom en gruppe personer og/eller organisasjoner om å utføre et bestemt arbeid, typisk til en gitt tid og til en gitt kostnad.
Ethvert prosjekt består av et hierarki av avtaler, og alle involverte parter må kjenne de avtaler som er inngått, og være trygge på at avtalene overholdes; dvs en avtaledisiplin må etableres. En avtaledisiplin forutsetter en plan for gjennomføring av avtalen, f.eks på hvilket ledelsesnivå bestemte typer avtaler må godkjennes, og denne planen må inneholde estimater for tid, kostnad, etc. Videre kreves kontrollrutiner for å sikre at avtalene overholdes. Det er den øverste ledelse som til sist er ansvarlig for å innarbeide god avtaledisiplin.

De viktigste elementer i å etablere en avtaledisiplin er:

- Gi tid til forberedelse og planlegging
- Klare ansvarsforhold
- Klar støtte fra ledelse
- Krav om oppfyllelse av avtaler

Det vil naturlig være et motsetningsforhold mellom prosjektplaner versus organisatoriske planer ordnet etter tidsperioder. Prosjektplaner definerer planer for de enkelte prosjekter, mens organisatoriske planer definerer den overordnede målsetning og sentrale prioriteringer for organisajonen som sådan.

Den øverste ledelse kan benytte kvartalsvise gjennomganger for å knytte sammen prosjektplanlegging og organisasjonsplanlegging. Kvartalsvise gjennomganger er et forum for å kontrollere status og fremdrift ved å vurdere prosjekter og organisasjonen som helhet opp mot sine planer.
Når nivå 3 er nådd kan fremdrift måles mot konkrete kvalitetsplaner, og når nivå 4 er nådd kan i tillegg kvantitative mål legges til grunn for vurderingene.

Disse gjennomgangene er også viktige for å løse konflikter, og det bør finnes et system for å behandle uenighet (contention system). Hensikten med dette er å oppmuntre til åpenhet om problemer for å finne fornuftige løsninger på de eksisterende problemer. For å kunne fatte gode beslutninger er det nødvendig med kunnskap om alle relevante sider av en sak, og ved å akseptere saklig uenighet vil de resulterende diskusjoner gi en bedre forståelse for problemene og derved mer kvalifiserte løsningsforslag.

Kvartalsvise gjennomganger fokuserer særlig på organisasjonen som helhet, og tiltak for prosessforbedringer er sentrale tema her.

I tillegg bør det finnes prosjektinspeksjoner for ulike prosjektfaser. Siste trinn i en prosjektinspeksjon er godkjennelse til å fortsette til neste fase. Den detaljerte definisjon av hver fase, og hvordan slike inspeksjoner skal utføres, er organisasjons- og prosjektavhengig.

Etablering av et ledelsessystem krever først godkjenning fra øverste ledelse. Deretter må ansvar tildeles og ressurser fordeles. Deretter følger:

- En avtaledisiplin etableres
- Rutiner for kvartalsvise gjennomganger defineres
- Rutiner for prosjektinspeksjoner ved angitte faser defineres

6. Prosjektplanen

6.1 Prosjektplanlegging

En prosjektplan definerer de overordnede oppgaver som skal utføres, samt gir et estimat for den tid og de ressurser som er påkrevd. Videre gir den et rammeverk som ledelsen kan benytte for kontroll og gjennomganger.

Prosjektplanen kan også være nyttig for læring som en benchmark for å sammenligne med den faktiske prosjektgjennomføring.

En prosjektplan utarbeides ved oppstart og modifiseres etterhvert. Modifikasjoner er nærmest alltid påkrevd ettersom de opprinnelige ressursestimater og tidsplaner viser seg for optimistiske. Dette skyldes igjen den læringsprosess man gjennomgår ved arbeid med prosjektet, både for kunde og leverandær, og den påvirkning et prosjekt har på den omgivelsen den skal virke i. Avvik fra opprinnelige estimater er derfor ikke nødvendigvis noens feil, men det må tas høyde for dette og man må være forberedt på det slik at det kan behandles som en naturlig del av prosjektet.

De sentrale elementer i en prosjektplan er:
- Kravspesifikasjon med motiv og målsetning
- Begrepsmessig design
- Work Breakdown Structure (WBS)
- Estimat av størrelse
- Estimat for ressursbruk
- Tidsplan

De sentrale elementer for en kravspesifikasjon er:
- Funksjonelle krav
- System behov (konfigurasjon, kompatibilitet, etc)
- Kundens behov og medvirkning
- Suksesskriterier
- Validering og akseptansekriterier
- Kundestøtte og vedlikehold

WBS splitter opp prosjektet i mindre enheter som kan betraktes separat m.h.p estimering av størrelse, ressursbruk, tidsplanlegging og gjerne også arbeidsdeling.

6.2 Måltall for Størrelse

Det er vanskelig å finne gode måltall for størrelsen på et prosjekt. Gode måltall bør være slik at de er relativt enkle å benytte tidlig i prosjektet, og lett tilgjengelige når arbeidet er ferdig.

'Managing the Software Process' diskuterer mye LOC (lines of code), hvor det forøvrig finnes ulike måter å foreta opptelling av dem. Det er imidlertid vanskelig å relatere LOC til tidlige funksjonelle kravspesifikasjoner.

Et alternativ er funksjonspunkter (function points). Dette tar utgangspunkt i brukerens perspektiv. Basert på kravspesifikasjonen vurderes kompleksiteten til en gitt funksjonalitet og den tildeles et gitt poengtall basert på erfaring. Ulempen med denne metoden er at den er basert på vurderinger, og opptellingen kan ikke automatiseres. Et alternativ er å benytte denne metoden for opprinnelige estimat, men deretter konvertere til LOC og deretter benytte LOC som automatisk kan måles.

Det finnes forøvrig mye litteratur av nyere dato om dette tema.

6.3 Estimering

Det er viktig å kunne gi rimelig gode estimater for prosjektstørrelse og ressursbruk ettersom dårlige estimat ofte medfører utilstrekkelige ressurser, tids- og kostnadsoverskridelser.

En metode som beskrives i boken er 'the Wideband Delphi Technique'. Her vil en gruppe av eksperter få programmets spesifikasjon. Basert på dette foretar de et estimat. Deretter sammenlignes alle estimatene, og anonymt foretar den enkelte justeringer inntil alle estimatene er innenfor gitte grenser.

Ved estimering er det normalt stor variasjon avhengig av type produkt, programstørrelse, ny kode versus endret kode, osv. Undersøkelser viser ofte at det er langt mer komplisert å endre kode enn å lage ny kode. Vedlikehold er igjen enda mer krevende ressursmessig enn videreutvikling.

Det er nødvendig å kalibrere alle data til den aktuelle organisasjon. De fleste faktorer som påvirker produktiviteten kan ikke fastsettes eller kompenseres for. Det er derfor alltid nødvendig å ta høyde for overskridelser. Boken bruker her 100%/200% i eksempler.

Det viktigste er uansett at estimater gjøres og registreres for senere sammenligning med oppnådd resultater. Estimering er en læringsprosess, og det viktigste for å kunne gi gode estimater er å opprette en egen arbeidsgruppe med dette som oppgave. Deres ansvar er å registrere data om estimater, foreta analyser av disse for å finne egnede mål på egne prosjekter, og assistere utviklingsgruppene når disse skal lage prosjektplaner.

Ferdigdefinerte kostnadsmodeller bør ikke benyttes ukritisk. Det er nødvendig å forstå prosjektet for å kunne gi gode estimater. God intuisjon om lokale forhold kombinert med dokumentert erfaring er normalt langt bedre, og dette gir også en læringseffekt som er viktig.

6.4 Prosjektkontroll

Prosjektplanen skal kunne benyttes av ledelsen for kontroll av status og fremdrift i henhold til planen.

Det er f.eks viktig ved fastsettelse av status at noe som er 'halvferdig' ikke teller med, men at man heller foretar en oppsplitting slik at kun ferdige oppgaver og aktiviteter telles med. Man har ikke arbeidet lenge med noe før det er 10% ferdig, og uansett reell status deretter er det gjerne 90% ferdig.

6.5 Utviklingsplan

Når prosjektplanen er ferdig vil den gjennomgå en på forhånd definert godkjennelsesprosess. Det er viktig at alle involvert får gått igjennom den slik at de kan godkjenne de avtaler som involverer dem selv.

7. Konfigurasjonsstyring - Del 1

Endringskontroll er en essensiell aktivitet i utviklingsprosessen. Konfigurasjonsstyring (Software configuration management SCM) er nødvendig for å kunne koordinere og kontrollere endringer foretatt av en mengde personer som samtidig arbeider på et prosjekt.

Dette kapittelet fokuserer på SCM for programkode, mens kapittel 12 også omfatter kravspesifikasjon, design, tester osv.

SCM skal holde rede på nåværende versjon, hvilke endringer er foretatt, hvordan endringer skal foretas, informere andre om konsekvenser for deres arbeid, osv.

En baseline er grunnlaget for konfigurasjonsstyring. Den er den offisielle nåværende versjon av prosjektet, og et lager for alt arbeid som er ferdig. Enhver etterfølgende endring betraktes som et inkrement av denne inntil ny baseline defineres. Hver versjon er entydig identifisert.

Utviklere sjekker-ut og sjekker-inn kode fra et sentralt lager som inneholder nåværende versjon, og hvor det er formelle rutiner for godkjennelse av endringer.

Formelle rutiner for endringskontroll omfatter registrering og godkjenning av endringsforslag før disse implementeres. Tilsvarende lagres ulike data om endringer som faktisk er foretatt. Helst slik at det vil være mulig å endre tilbake til opprinnelig status dersom endringen viser seg uheldig i ettertid.

Følgende er 3 viktige roller for konfigurasjonsstyring:
- Configuration manager
- Moduleier
- Change Control Board (CCB)

Configuration manager er ansvarlig for følgende:
- Utvikle, dokumentere og distribuere prosedyrer for SCM
- Etablere baseline for prosjektet
- Sikre mot uautoriserte endringer i baseline
- Sikre at alle endringer i baseline er tilstrekkelig dokumentert (slik at kan reproduseres eller trekkes ut).
- Sikre at alle endringer i baseline er regresjonstestet

Det kan være nyttig for kontroll å tilordne enhver modul en gitt utvikler som eier/ansvarlig for denne. Moduleier skal fullt ut forstå modulens design, ha ansvar for endringer og sikre at regresjonstesting utføres når påkrevd.

Ved større prosjekter kan det være behov for en sentral kontrollmekanisme som sikrer overordnet kontroll og koordinasjon av endringer. Dette er oppgavene til CCB. Deres ansvar er å forsikre seg om at alle endringer i baseline er korrekt utført, at alle utviklere involvert er informert om endringer, og at alle endringer er formelt godkjent.
Det er CCB som gjennomgår anmodninger om endringer (change requests) og godkjenner disse, avviser dem, eller bestemmer å utsette dem inntil videre.

Systemet for endringskontroll registrerer anmodninger om endringer, rapporter om problemer, CCB aksjoner, og en logg for alle aktiviteter.

Konfigurasjonsstyring kan være meget tidkrevende, og under press kan det være fristende å slakke på krav til formelle inspeksjoner og godkjenninger. Det er imidlertid under press at feil grunnet dårlig konfigurasjonsstyring oftest oppstår.

Endringsaktivitet kan være viktige indikatorer for status til et prosjekt. Irregulære mønstre her kan tyde på problemer. Aktivitetsloggen for endringer er derfor viktige styrings- og kontrolldata for prosjekter.

8. The Software Quality Assurance Group (SQA)

Software Quality Assurance (SQA) er en arbeidsgruppe eller avdeling hvis rolle er å forsikre ledelsen om at utviklingsarbeidet utføres slik det skal i henhold til standarder, prosedyrer, osv, som er fastsatt for organisasjonen. Hvis ikke er de ansvarlig for å varsle ledelsen om dette. SQA er imidlertid ikke selv ansvarlig for kvalitetsmål i produktene eller for å lage kvalitetsplaner.

Før opprettelse av en SQA funksjon er det viktig å få definert et rammeverk for kvalitetskrav; dvs hvor viktig er kvalitet i organisasjonen? Skal f.eks en leveranse utsettes dersom ikke kvaliteten er tilstrekkelig (i.h.t de krav som er definert for dette).

Er det nødvendig med en separat SQA funksjon? I mindre bedrifter kan kanskje ledelsen selv stå for dette. Alternativt kan utviklere fylle denne rollen for hverandre, men erfaring viser at dette ikke fungerer for større organisasjoner.

For å være effektive bør følgende være oppfylt:
- SQA rapporterer til ledelsen via en egen kanal.
- SQA består av kompetente ansatte (grunnet en ofte utakknemlig jobb vil det ikke alltid være tilfellet).
- SQA ser sin rolle som en støttefunksjon til utvikling og vedlikehold.
- SQA har full støtte fra øverste ledelse (i det motsetningsforhold som lett oppstår i forhold til utvikling og vedlikehold).

SQA må rapportere på et tilstrekkelig høyt ledelsesnivå til at de får gjennomslag for sine krav og prioriteringer. Det er imidlertid også en avveining mot at desto lavere nivå de rapporterer til desto bedre vil ofte forholdet være til utviklingsavdelingen. Dette er organisasjonsavhengig. Generelt er støtte fra ledelsen avgjørende for en effektiv SQA funksjon.

Ansatte i SQA må være kjent med statistiske metoder, prinsipper for kvalitetskontroll, utviklingsprosessen, og evnen til å samarbeide konstruktivt med mennesker i stressende situasjoner.

SQA funksjonen må selv evalueres m.h.p effektivitet. Dette kan gjøres ved å vurdere kvalitet på leveranser i ettertid og sammenligne med foregående evalueringer fra SQA. Alternativt bør det også foretas formelle inspeksjoner av arbeidet til SQA.

Hvert prosjekt bør ha en SQA plan (SQAP) som spesifiserer målsetning, SQA oppgaver som skal utføres, standarder og prosedyrer som det skal kontrolleres mot. Mengden av arbeid for SQA kan ofte bli så stort at kontroll foretas mot et statistisk valgt utsnitt.

Følgende er noen vanlige årsaker til at SQA ikke utfører sitt arbeid som ønsket:
- De ansatte har ikke tilstrekkelig kompetanse til denne jobben.
- Samarbeidsforholdet til utvikling er meget dårlig
- Øverste ledelse støtter oftest utvikling ved konflikter
- SQA har ikke tilstrekkelig med dokumenterte og godkjente standarder og prosedyrer.
- Utviklingsgruppene har ikke verifiserbare kvalitetsplaner. Dette medfører typisk vurderinger av antall feil o.l fremfor på den overordnede kvalitet og indikatorer på overordnet kvalitet.

En av de vanskeligste oppgavene er å få dyktige personer til å arbeide i SQA. En metode er å sørge for at utviklingsledere alltid skal rekrutteres via SQA.

Del 3 : Den Definerte Prosessen

9. Standarder og Prosedyrer

9.1 Hensikten med Standarder

Standarder og prosedyrer beskriver egenskapene til et objekt, eller hvordan en aktivitet skal utføres. De gir grunnlag for sammenligning av egenskapene til objekter og aktiviteter; f.eks størrelse, innhold, kvalitet, etc.
For eksempel, en standard for inspeksjon av programvare definerer gjerne materiale til forberedelser, deltagere, ansvarsområder, resultatdata, osv, mens en tilhørende prosedyre definerer hvordan arbeidet skal utføres, av hvem, når, og hva som gjøres med resultatet.

Standarder og prosedyrer fremmer konsistent bruk av verktøy og metoder, og de er nødvendige for SQA i deres arbeid. De gjør det lettere for ansatte å flytte mellom prosjekter, reduserer tid til opplæring, gjør det enklere å inspisere og gjennomgå hverandres arbeid. Det er også noe alle kan kommunisere om og arbeide mot.
For mye dokumentasjon kan være sterkt hemmende, men det er tross alt enklere å påpeke overflødige dokumentasjonskrav enn å kunne være presis m.h.p manglende dokumentasjon. Fordel med standardiserte, dokumenterte prosedyrer er at de er håndfaste. Kan kommenteres og kritiseres. Svakheter kan enklere avsløres, særlig om noenlunde formelt beskrevet.

9.2 Etablering av Stadarder

Før det etableres nye standarder bør det utarbeides en plan som vurderer følgende momenter:
- Tilgjengelige standarder
- Organisasjonens behov etter prioritet
- Status på dagens prosjekter
- Ansattes tilgjengelighet og kompetanse
- Hvordan sikre bruk av de standarder som eventuelt defineres

Standarder og prosedyrer kan gjerne kategoriseres som støtte for følgende:
- Ledelse og planlegging
- Utvikling
- Prosess og verktøy

Arbeidet med å utvikle en standard bør utføres av enkeltindivider eller mindre grupper, helst med representanter fra gruppene som vil implementere og sikre bruk av standardene.

Det er passende med en standard dersom det ikke er påkrevd med overveininger og vurderinger. Dersom det er usikkert om en standard er passende kan det alternativt lages retningslinjer, eller standarden kan defineres slik at den gir rom for individuelle tilpasninger eller unntak i visse situasjoner.
Dersom en eksisterende standard overtas bør denne nøye tilpasses den lokale organisasjonen.

Gjennomgang av standarder under utvikling har både en ledelsesmessig og teknisk funksjon, og dette bør inkludere alle deler av organisasjonen for kommentarer. I tillegg bør formelle inspeksjoner benyttes (se kap.10). Før en standard tas i bruk er det påkrevd at den testes; tilsvarende andre komponenter i et programvareprodukt.

Standarder må vedlikeholdes og holdes oppdatert, og de må derfor tilpasses basert på erfaring ved bruk av dem, og erfaring fra det å sikre at de overholdes. Det bør imidlertid ikke være særlig tidkrevende å vedlikeholde en standard. Dersom mange endringer er påkrevd tyder det på at det aktuelle området ikke er klart for standardisering enda.

SQA har som et av sine hovedansvarsområdet å sikre at standarder brukes slik de er ment brukt. Dette gjøres ved en kombinasjon av inspeksjoner og tester. Noen ganger kan det lages automatiserte verktøy som benyttes for å kontrollere at standarder overholdes.

10. Formelle Inspeksjoner og Gjennomganger

10.1 Ulike typer inspeksjoner

En formell programvareinspeksjon eller -gjennomgang (software inspection/review) er en inspeksjon av en programmerer's arbeid av andre kollegaer (peer review) med det mål å avdekke problemer og å forbedre kvaliteten på arbeidet. Fordelen med slike inspeksjoner er at de kan utføres tidlig i utviklingsprosessen.

Det er normalt vanskelig for en person selv å identifisere mange feil og svakheter i sitt eget produkt. Dette ikke fordi man ønsker å tilsløre feil, men fordi man allerede er bundet opp i et gitt syn på det man arbeider med.

Det finnes mange typer inspeksjoner, f.eks:
- Management review
- Technical review
- Software inspection
- Walkthrough

Management og Technical review er primært rettet mot ledelsen for å gjennomgå fremdrift og at planer og spesifikasjoner følges. Software inspection og walkthrough er rettet mot å avdekke feil, men hvor sistnevnte er mer uformell og med læring som mål.

Inspeksjoner er kostbare og bør derfor begrenses til der hvor det er sannsynlig at det er behov for dem. Software inspections er særlig nyttige for design og kode, men gjerne også for kravspesifikasjoner, tester og dokumentasjon. Technical reviews bør brukes for utviklingsplaner og testplaner.

I tillegg til å avdekke feil og problemer er inspeksjoner også viktige for å verifisere at et arbeid tilfredsstiller visse predefinerte kriterier; f.eks standarder og prosedyrer. En viktig bieffekt er at ansatte får kunnskaper om produktet som gjennomgås.

10.2 Gjennomføring av inspeksjoner

Følgende er noen viktige prinsipper for formelle inspeksjoner:
- Inspeksjonen er en formell prosess med sjekklister og definerte roller for deltagerne.
- Generiske sjekklister og standarder er utviklet for ulike typer inspeksjoner, og individuelt tilpasset den enkelte inspeksjon.
- Deltagerne er forberedt og har derfor spørsmål o.l klar før inspeksjonen. Forberedelse er essensielt, og normalt vil ca.75% av feil og mangler avdekkes under forberedelsene.
- Inspeksjoner skal fokusere på å identifisere problemer, ikke på å løse dem.
- Resultatene presenteres skriftlig.

Det er generelt viktig under inspeksjoner at det fokuseres på den totale kvaliteten og ikke bare på antall feil avdekket. Dersom det er for mange feil på detaljnivå er det lett å overse mer alvorlige problemer.

Inspeksjoner må planlegges og de skal være en eksplisitt del av enhver prosjektplan. De bør videre utføres før enhver større eller mindre milepæl; f.eks leveranse av hele eller deler av et produkt.

Resultatene fra inspeksjoner skal fokusere på evaluering av produktet, status, fremdrift, osv, og ikke som en evaluering av personer ansvarlige for produktet.

Det er viktig å foreta formell rapportering fra inspeksjoner både for å sikre at de utføres som de skal, og fordi resultatene kan benyttes til å måle effektiviteten på inspeksjonene som er holdt. Sistnevnte er viktig for kontinuerlig forbedring av utviklingsprosessen (nivå 5).

Følgende er vanlige årsaker til at inspeksjoner blir lite effektive:
- Utilstrekkelige forberedelser
- For mye materiale ble gjennomgått
- For mange deltagere

- Feil personer deltok

Undersøkelser viser at formelle inspeksjoner er meget effektive til å forbedre kvalitet og produktivitet. De er normalt langt mer effektive enn tester, og i tillegg avdekker de feil og problemer tidligere i utviklingsprosessen. Dette overflødiggjør ikke testing, men formelle inspeksjoner bør prioriteres relativt.

11.Testing av Programvare

11.1 Prinsipper for Testing

Pareto prinsippet - 20% av modulene innholder 80% av feilene.

Det sentrale mål for testing er å avdekke hvorvidt en modul (eller testenhet) tilhører de 20% av problemtilfeller (en 'jungel') eller om den tilhører de restende som er lite problematiske (en 'hage med litt ugress'). Dersom en modul er en 'jungel' er det ikke tilstrekkelig å fjerne feil ved testing. Det kan da ofte være nødvendig med redesign eller omprogrammering, mens dersom den er en 'hage med litt ugress' vil testing kunne fortsette for å luke ut flest mulig av de gjenværende feil. Testing er m.a.o viktig som en kvalitetsindikator.

En god test er en test som har stor sannsynlighet for å avdekke feil, ikke en test som viser at programmet virker. Tester kan uansett ikke vise feilfrihet.

Testing er generelt lite effektive i å avdekke feil, f.eks sammenlignet med inspeksjoner. Testing kan normalt bare utføres på et lite antall av de mulige paths som et program kan gjennomgå ved runtime. Denne svakheten vil antagelig øke i tiden fremover ettersom programvare øker i kompleksitet og det blir derfor stadig vanskeligere å inkludere for testing et tilstrekkelig antall tilfeller til at mulighetene for dynamisk oppførsel til systemet er dekket i rimelig grad (f.eks ved komponent-basert utvikling).

Testmetoder:

- *White-box testing*
  Tester som tar hensyn til den interne struktur i modulen som testes.

- *Black-box testing*
  Tester som ikke tar hensyn til intern struktur, men kun grensesnittet utad for modulen. Vanligvis basert på de funksjonelle krav til modulen.

Tester kan klassifiseres til følgende typer (det finnes noe ulike klassifikasjoner i litteraturen):

- *Enhetstest*
  Testing av enkeltprogrammer eller enheter i simulerte testomgivelser. Bruker ofte white-box metoden.

- *Funksjonstest*
  Tester eksterne funksjonskrav, typisk i.h.t kravspesifikasjonen. Bruker black-box metoden.

- *Integrasjonstest*
  Integrasjonstesting er en bro mellom funksjonell black-box testing og white-box testing av individuelle enheter. Vil gjerne utføres enten top-down, bottom-up, eller som en mellomting. Begge deler har sine fordeler og ulemper m.h.p tilleggsprogrammer (drivere og scripts) som må lages for å kunne foreta tester underveis i inkrementell integrasjon.

- *Regresjonstest*
  Utfører tidligere tester, funksjons- og integrasjonstester, for å sikre at ingen nye feil etter endringer. En metode er å kjøre delvise regresjonstester (dvs inkludert et utvalg av moduler endret) ofte, og komplette regresjonstester før viktige milepæler og leveranser.

- *Systemtest*
  Tester hele systemet mot kravspesifikasjonen. Dette enten i en simulert omgivelse hos utvikler, eller også i den reelle omgivelse hos kunden. Dette inkluderer også sikkerhetsaspekter, brukervennlighet, ytelse, osv, dvs ønsker som kanskje ikke er kvantifisert i kravspesifikasjon.

- *Akseptansetest*
  Mye tilsvarende systemtest, men skal utføres hos kunden i de reelle omgivelser under bruk (alternativt betatesting dersom 'hyllevare'). Skal undersøke hvordan tilfredsstiller kundens krav totalt sett, uavhengig av opprinnelig kravspesifikasjon.

En test må betraktes som et eksperiment, og dette krever en hypotese som testen skal bevise eller motbevise. Følgende er noen prinsipper for tester:

- En god test er en test som har en høy sannsynlighet for å avdekke feil, ikke en test som viser at programmet virker.
- Det er viktig å definere kriterier for når testing kan avsluttes
- Utviklere bør ikke teste sine egne programmer
- Ettersom antall feil oppdaget øker vil også sannsynligheten for flere feil øke.

For å kunne bedømme en test m.h.p sannsynligheten for å avdekke feil er det nødvendig med måltall som viser hvordan en test har fungert tidligere; dvs vurdering av testers effektivitet. Dette er tema på nivå 4.

Det kan være vanskelig å definere kriterier for når testing er ferdig. Spørsmålet er da ikke om alle feil er funnet, men om programmet har tilstrekkelig kvalitet til at testing kan avsluttes. I mangel på data til å kunne gjøre fornuftige valg er det oftest slik at testing pågår inntil tid avsatt til dette er oppbrukt.

Utover enhetstest bør ikke utviklere teste egne programmer. Ikke fordi de vil underslå testresultater, men fordi de vil være så influeret av sine forutinntatte oppfatninger av programmet at de lettere vil overse feilkilder.

11.2 Testplanlegging

Testplanlegging starter med en overordnet prosjektplan som definerer funksjoner, roller og metoder for alle testfaser. Boken inneholder eksempler på sjekklister for testplanlegging, og følgende er sentrale punkter i en testplan:

- Beskrive målsetning for hver testfase
- Tidsplaner og hvem ansvarlig for hver testaktivitet
- Beskrive verktøy og andre fasiliteter som trengs
- Beskrive standarder og prosedyrer for planlegging, utførelse og rapportering fra testingen.
- Definere kriterier for når testing er ferdig, og suksesskriterier for hver test.
  Dette er vanskelig, men generelt vil mål for testere være å avdekke flest mulig feil, mens mål for utviklere er å produsere programmer med færrest mulig feil.

Det er viktig at tester planlegges allerede under kravspesifikasjonsfasen, og at det her tas høyde for hvordan ulike elementer i denne kan testes.

Det største problem med design av tester er å velge mellom dem. Det er normalt ikke mulig å foreta komplett testing. Det finnes forøvrig mye litteratur om testing, og hvordan tester skal designes for å dekke flest mulig tilfeller. Det beste er imidlertid om data om testeffektivitet er tilgjengelig slik at det kan prioriteres mellom ulike tester og testtyper.

Det er viktig at testresultater oppbevares for senere bruk, både for å kunne reprodusere tester, og for analyse, måling av testeffektivitet, osv. Testdata er generelt en viktig ressurs for prosessforbedring. Boken gir eksempler på data som typisk kan være ønskelig å registrere for tester. Ettersom det er et utall muligheter er det viktig å foreta et velbegrunnet utvalg.

12. Konfigurasjonsstyring - Del 2

Dette kapittelet utvider SCM fra kap.7 til også å omfatte andre prosjekt elementer enn kildekode, dvs kravspesifikasjon, design, osv. Det er også aktuelt å inkludere operasjonelle data samt for verktøy, f.eks kompilatorer o.l, som benyttes. Totalt skal SCM omfatte alle komponenter som produseres eller benyttes under prosjektets gang.

Første trinn når skal opprette et SCM system er å utvikle en konfigurasjonsstyringsplan (SCMP) som inneholder målsetning, ansvarsfordeling, og hvilke metoder som skal brukes. Denne er også grunnlaget for evaluering av SCM for å sikre at standarder og prosedyrer for dette følges.

SCM krever entydig identifikasjon av de ulike SCI (Software Configuration Items) involvert. Det må derfor defineres et system for entydig navngiving av de ulike items involvert. Denne navngiving må også underlegges SCM og er en viktig del av planlegging av SCM.

Forøvrig som beskrevet i kap.7, det sentrale for SCM er systemet som holder rede på baselines, metoden med låsing og inn/ut-sjekking for individuelt arbeid, samt rutiner for formell vurdering av endringer og godkjenning og oppfølging av disse. Det opprettes gjerne separate baselines for ulike typer SCI; f.eks kravspesifikasjon, design, testing, osv.

Feilrapportering er viktig og betraktes ofte som en del av SCM. Dette omfatter:

- Feilrapporter og rutiner for logging av feil
- Registrering av reparerte feil
- Rapportering av status m.h.p feil

13. Modeller av Utviklingsprosessen

En prosessmodell definerer livssyklusen for utviklingsprosessen, dvs tekniske og ledelsesmessige aktiviteter som utføres underveis. Prosessen dekomponeres til aktiviteter og oppgaver som hver har et sett av forbetingelser, en funksjonell beskrivelse, verifikasjons- og testprosedyrer, kriterier for når er ferdig, osv.

En dokumentert utviklingsprosess gir organisasjonen et konsistent rammeverk med retningslinjer for hvordan prosessen skal utføres. Prosessmodellen må imidlertid tilpasses organisasjonens lokale forhold, og den vil også variere noe avhengig av størrelse, betydning og tekniske egenskaper til et bestemt utviklingsprosjekt.

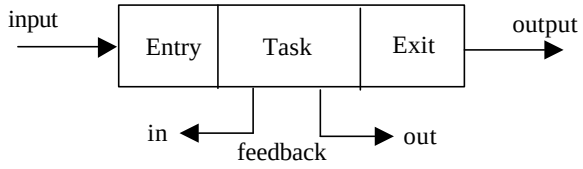
'Managing the Software Process' definerer 3 ulike abstraksjonsnivåer som prosessmodeller kan betraktes fra.

- Universal(U) nivået spesifiserer typisk de overordnede retningslinjer (policies) for organisasjonen.
- Worldly(W) nivået spesifiserer typisk prosedyrer og metoder som skal brukes i det daglige utviklingsarbeidet.
- Atomic(A) nivået spesifiserer typisk de standarder som skal benyttes på W nivået.

De fleste prosessmodeller er noenlunde like som sett fra U nivået. Først på W nivået vil til dels store forskjeller komme frem, men mange arbeidsoppgaver på W nivået er gjerne standardisert over flere prosjekter. De benyttes kanskje i ulike sekvenser og kombinasjoner av A nivå standarder.

Det finnes en rekke overordnede prosessmodeller (U-nivået) som f.eks fossefallsmodellen, spiralmodellen, rapid application development (RAD), inkrementelle utviklingsmodeller, etc. De gir gjerne et representativt bilde av den overordnede arbeidsflyt, og kan være nyttige som beskrivende modeller for forståelse (deskriptive), men de er ikke tilstrekkelig detaljerte til å fungere som gode rammeverk for veiledning i utviklingsprosessen (ikke preskriptive).

'Managing the Software Process' beskriver en metode for utvikling av prosessmodeller som tar utgangspunkt i modellkomponenter (se figur nedenfor) som angir forbetingelser og input-verdier ved oppstart; standarder, prosedyrer, metoder, ansvarsforhold, etc, under selve aktiviteten; og hvilke resultater som skal produseres, validering av disse, osv. I tillegg angis interaksjon (feedback) til og fra andre aktiviteter under utførelsen.



Slike prosessmodellkomponenter kan utvikles og benyttes over flere prosjekter hvor de for hvert prosjekt konfigureres og eventuelt detaljeres i.h.t de individuelle egenskaper til prosjektet, eller eventuelt grupper av relaterte prosjekter. Motsetningsforholdet mellom lokale behov og standardisering løses ved generiske prosessbeskrivelser som kan konfigureres på ulike måter, og som kan utvides og modifiseres lokalt.

Utvikling av en prosessmodell kan på mange måter betraktes som et ordinært utviklingsprosjekt. Modell-komponentene kan betraktes som objekter med egne livsløp og tilstandsgjennomganger over tid; dvs de kan betraktes som såkalt 'aktive objekter' i objekt-orientert terminologi.

14. The Software Engineering Process Group (SEPG)

For at det skal bli en forbedring av programvareutviklingsprosessen må noen gis ansvaret for at jobben blir utført. For å oppnå resultater med prosessforbedring må det selv betraktes som et utviklingsprosjekt, dvs:

- Identifiser de viktigste problemer
- Etabler prioriteter
- Lag handlingsplaner
- Fordel mennesker og ressurser
- Gjør øvelser
- Start implementasjon
- Kontroller fremdrift og status

Software Engineering Process Group (SEPG) tilordnes det daglige ansvar for forbedringsprosessen, dvs de skal:

- Initsiere og drive forbedringsprosessen
- Etablere prosesstandarder
- Vedlikehold prosessdatabasen
- Spre kunnskap om prosessen
- Støtte utviklingsgruppene m.h.p endringer innført, og i bruk av nye metoder, standarder, ny teknologi.
- Foreta periodevise status rapporter og vurderinger av prosessen

SEPG må være kjent med status for utviklingsprosessen, identifisere hvor forbedring kan gjøres, og gi praktiske råd til ledelsen om nye forbedringstiltak. SEPG har ansvar for at endringer implementeres effektivt og uten unødige forstyrrelser for det daglige utviklingsarbeid.

SEPG bør ha en separat kanal for rapportering til ledelsen, f.eks via samme kanal som SQA, men ikke til SQA da SEPG skal etterstrebe nøytralitet i det naturlige motsetningsforholdet mellom utvikling og SQA.

SEPG bør ikke selv være en støttefunksjon for teknologibruk og introduksjon. Det bør finnes en egen gruppe for dette, men SEPG kan assistere i dette. Tilsvarende bør det finnes en egen gruppe for undervisning og opplæring, men SEPG kan stå for initsiell opplæring av de som er ansvarlig for opplæring.

Initsielt bør SEPG fokusere på prosjektplanlegging og prosjektledelse. Det viktigste er imidlertid å begrense fokus til oppgaver som kan utføres relativt hurtig og effektivt. Den største fallgruben er å forsøke å rekke over for mye slik at lite blir gjort.

SEPG har antagelig ikke selv ressurser til å lage standardene, men de har ansvar for at dette arbeidet utføres; dvs:

- SEPG fastsetter prioriteter for utvikling av standarder
- Forsikre at tidligere relevant arbeid for en standard vurderes
- Opprette arbeidsgrupper som skal definere standarden
- Formell inspeksjon og godkjenning av standarder

SEPG har ansvar for vedlikehold av *prosessdatabasen*.
Prosessdatabasen er et lager for data samlet om utviklingsprosessen og produkter produsert; dvs data om størrelse, kostnader og tidsplaner, måltall for produkter samt måltall for prosessen. Disse data gir grunnlag for å forbedre estimater, analyser og vurdering av produktkvalitet.

Følgende er viktige kriterier for en effektiv prosessdatabase:

- De data som samles inn må være velbegrunnet. Ellers kan det være et tidssluk til liten nytte.
- Data må være presist definert m.h.p format o.l.
- Data må kunne samles inn relativt enkelt, i tide til sitt bruk, og de må være tilstrekkelig komplette
- Data må verifiseres
- Det må finnes rutiner for hvordan data skal registreres, og det må være satt av ressurser til dette.
- Data må beskyttes og vedlikeholdes, og det må finnes rutiner for aksess til data for analyse o.l.

SEPG kan evalueres ved å undersøke i hvilken grad de selv anvender standarder og metoder for prosess-forbedring. SEPG bør normalt være 1 nivå høyere i modenhetsmodellen enn resten av organisasjonen.

Del 4 : Den Kontrollerte Prosessen

15. Datainnsamling og Analyse

15.1 Prinsipper for Datainnsamling

En prosess sies å være under 'statistisk kontroll' dersom dens resultater kan forutsies innen visse statistisk definerte grenser. Når en prosess er under statistisk kontroll betyr det at dersom den gjentas på noenlunde samme måte vil resultatet også bli noenlunde det samme.

Det grunnleggende prinsipp bak statistisk kontroll er målinger. Måltallene må representere prosessen tilstrekkelig presist, og de må være tilstrekkelig veldefinert og verifisert til at de gir et pålitelig grunnlag for forbedringstiltak.

Data om utviklingsprosessen er viktige for å lære hvordan denne kan forbedres. For eksempel, kvalitet er også et spørsmål om økonomi. Det er alltid mulig å utføre flere inspeksjoner, flere tester, osv. Ved kvantifisering av prosessen finnes etterhvert data tilgjengelig for bedre å kunne vurdere kost-nytte effekten av det arbeid som gjøres og de prioriteringer som må foretas.

Følgende er noen prinsipper for effektiv datainnsamling:

- Data samles inn i henhold til en plan med en spesifikk målsetning.

  Data kan ikke samles inn tilfeldig. Det må ligge en plan bak.

- Valg av data som samles inn er basert på en modell eller hypotese om prosessen som måles.

  For effektiv måling må man kjenne de forventede resultater. Prosessen må derfor være forstått. For eksempel, gitt tilstrekkelig med målinger kan man alltid finne tilsynelatende statistisk signifikante resultater og korrelasjoner fra tilfeldige hendelser. Disse kan ikke brukes til forutsigelser, og en årsak-virkning forbindelse er derfor påkrevd. Det er derfor viktig at måltall kan knyttes til bestemte deler av en prosessmodell; dvs prosessmodellen må være på plass for effektiv måling.

- Effekten av datainnsamling på organisasjonen og dens ansatte må vurderes.

  Man må vurdere hvordan målinger påvirker mennesker, og hvordan mennesker påvirker målinger.

  Ansatte vil typisk gjøre alt for at målingene skal bli bedre, nesten uansett alt annet. Det er derfor avgjørende at forbedring av måltall korresponderer med en forbedring av prosessen.

  Målinger må ikke være truende eller evaluere de ansatte. Det viktigste er å vise hvordan målinger kan forbedre arbeidssituasjonen for den enkelte; dvs motivere for målinger.

  For å sikre objektivitet og effektivitet er det ønskelig å kunne automatisere så mye som mulig av målingsprosessen. Det bør derfor være mulig å samle inn data direkte fra utviklingsprosessen.

- Planen for datainnsamling må ha ledelsens støtte.

  Ettersom datainnsamling både er kostbart og det normalt er en relativt stor tidsforsinkelse mellom når målinger foretas og når gevinst oppnås må dette betraktes som en langsiktig investering.

Datainnsamling må planlegges, og data innsamlet må valideres for å sikre at de er representative for prosessen.

Planen må spesifisere:

- Hvem skal bruke data
- Hvordan vil data bli brukt
- Hvorfor er disse data nødvendige
- Speifikasjon av data, format o.l
- Hvem vil samle inn data

- Hvordan vil data samles inn
- Hvordan skal data valideres

15.2 Måltall for Programvare

Der finnes mye litteratur om dette. Følgende er noen egenskaper som er ønskelige:

- Måltallene må være robuste; dvs repeterbare, presise og relativt ufølsomme for mindre endringer i verktøy, metoder eller produkt karakteristika.
- Måltallene må foreslå en norm; f.eks desto lavere verdi desto bedre.
- Måltallene bør relateres til spesifikke produkt eller prosess egenskaper.
- Måltallene bør følge naturlig fra prosessen; dvs som et bi-produkt fra prosessen slik at enklere å utlede dem.
- Måltallene bør være enkle å analysere.
- Måltallene bør være forutsigbare og kontrollerbare; dvs slik at kan gjøre prediksjoner som deretter sammenlignes med det faktiske resultat.

Måltall kan også klassifiseres som følger:

- Objektive vs subjektive
- Absolutte vs relative
  Absolutte er uavhengige av f.eks utvidelser i prosjektet.
- Eksplisitte vs utledete
- Dynamiske vs statiske
  Dynamiske måltall er tidsavhengige.
- Prediktive vs forklarende
  Prediktive måltall kan finnes eller lages på forhånd, mens forklarende først finnes i ettertid.

Målsetningen bør være å helst benytte objektive, absolutte, eksplisitte og dynamiske måltall for å kontrollere og forbedre en prosess.

'Managing the Software Process' fokuserer mye på LOC, dvs antall linjer kildekode, for måling av størrelse, som igjen er et viktig grunnlag for f.eks tidsestimater ved LOC pr.mnd for utviklere. Et annet måltall er antall feil i programmer som er mye benyttet i eksempler.

Uansett, det er viktig for en organisasjon å selv finne egnede måltall basert på lokale forhold, og den eneste måten å lære hvordan samle inn data er å begynne å samle inn data.

15.3 Dataanalyse

Boken gir en rekke eksempler på analyser av måltall for å vurdere f.eks effektivitet av inspeksjoner eller tester, evaluering av SQA, osv.

Testing er den fasen i utviklingsprosessen hvor det kanskje er enklest å samle data, og derfor kanskje det beste stedet å starte med datainnsamling og analyse.

16. Kvalitetssikring og Kvalitetskontroll

16.1 Måltall for Kvalitet

Produktkvalitet er antagelig det beste mål på et utviklingsprosjekt. Kvalitetsmål gir et klart mål på fremdrift, og de gir grunnlag for å definere målsetninger.

Hensikten med kvalitetsmålinger er å motivere til handling. For å sikre at disse handlingene resulterer i bedre kvalitet på produktene er det viktig at målingene er representative for kundens oppfatning av god kvalitet.

Dersom det ikke defineres noen kvantitative kvalitetstall , f.eks feil funnet i ulike faser i prosessen, effektivitet av inspeksjoner, osv, er det kun tids- og kostnadsplanene som representerer et kvantitativt mål. Det at en tidsplan følges kan imidlertid ikke uten videre relateres til nivået av produktkvalitet.

Det bør ikke defineres for få kvalitetsmål, isåfall vil alt fokus rettes mot disse og det finnes ingen måltall som alene kan karakterisere god kvalitet. Det bør heller ikke være for mange, men et mindre antall bør velges som tilsammen gir en god indikasjon på overordnet kvalitet.

Kvalitetsmålinger kan karakteriseres ved hvorvidt de er:
- Objektive: Kan de repeteres av andre?
- Tilgjengelide: Er det tilgjengelige i tide til å påvirke prosessen?
- Representative: I hvilken grad reflekterer de kundens syn på god kvalitet?
- Kontrollerbare: I hvilken grad kan utvikling eller vedlikehold påvirke verdien?

Ønskede måltall er de som er tilgjengelige i tide, enkle å samle inn, representative for kundens ønsker og behov, og direkte kontrollerbare under utvikling og vedlikehold.
Det finnes imidlertid ingen måltall som tilfredsstiller alt dette. For eksempel, antall feil er tilgjengelig tidlig i prosjektet, noenlunde objektivt, kontrollerbart, men de reflekterer ikke direkte kundens oppfatning av kvalitet. Målinger som derimot reflekterer kundens syn på kvalitet er ofte ikke tilgjengelige før sent i prosessen, gjerne også kostbare å samle inn (f.eks ytelse) eller subjektive (f.eks brukervennlighet). Disse er imidlertid viktige for bl.a akseptansebeslutninger. Generelt, når man skal bestemme hvilke måltall å benytte er det viktig å vurdere målsetningen med kvalitetssikringen.

Følgende er eksempler på måltall for kvalitet:
- Antall feil
  Over et visst nivå vil ikke dette være mål på kvalitetsoppfatning hos kunden.

- Endringsaktivitet
  Kan være nyttig som indikator på kvalitetsproblemer, men kan ikke alltid kontrolleres av utviklere dersom f.eks skyldes endringer i kravspesifikasjon.

- Kompleksitetsmål på programvarestruktur

- Test av operasjonell egenskaper i simulert omgivelse

- Antall brukerrapporterte problemer
  Er moderat relatert til kundes oppfatning av kvalitet, og er i tillegg subjektive og ikke tilgjengelige før etter levering.

- Installasjonsegenskaper
  Hvor enkelt eller vanskelig det er å installere.

- Spørreundersøkelser blant kunder
  Ikke tilgjengelig før lang tid etter levering.

- Pålitelighet og tilgjengelighet
  Kan vanskelig forutsies før operasjonell testing, og da noen ganger i komplette omgivelser for bruk.

Ferdig definerte matematiske modeller for estimering er ofte ikke tilstrekkelig gode ved at de ikke tar tilstrekkelig hensyn til de individuelle karakteristika til et prosjekt, og derfor ofte legger til restriksjoner som ikke er reelle. Dette er noe menneskene involvert i prosjektet bedre vil være istand til å bedømme.

16.2 Etablering av Kvalitetssikring

Etablering av kvalitetssikring har mye til felles med prosjektplanlegging som beskrevet i kap.6.
En kvalitetsplan utarbeides ved start av prosjektet. Ledelsen må involvere seg, definer en målsetning, lag en plan med kvantifiserbare måltall, kontroller fremdrift og resultater, juster planen. Før godkjenning vil planen sammenlignes med organisasjonens definerte kvalitetsmål.

Hensikten med en kvalitetsplan er å motivere til handling, ikke for å evaluere ansatte. Dersom målsetningen ikke nås vil dette gi opphav til handlingsplaner for å iverksette forbedringer. Dersom alle målsetninger alltid nås kan det indikere en dårlig kvalitetsplan da det er alltid rom for forbedring.

Del 5 : Den Optimaliserbare Prosessen'

17. Å Forebygge Feil

Å finne og reparere feil står for en stor andel av kostnadene ved programvareutvikling. Ettersom programvareprosjekter blir større og større, og kompleksiteten i produktene stadig øker, vil testing bli en stadig mindre effektiv metode for å avdekke feil. Andre metoder som formelle inspeksjoner og forebygging av feil blir derfor stadig viktigere.

Det er mer lønnsomt å kunne forhindre feil enn å oppdage og rette feil i ettertid. Å forhindre at feil oppstår i utgangspunktet (defect prevention) er derfor en viktig teknikk for prosessforbedring. Effekten er gjerne selv-forsterkende ved at dersom feil forhindres vil inspeksjoner og tester bli mer effektive m.h.p å finne de resterende feil. Det gis imidlertid ofte mer kreditt for å løse et problem enn for å hindre at problemet oppstår i utgangspunktet. Det gis derfor ikke alltid tilstrekkelig kreditt for problem forebygging. Fra nivå 4 er det imidlertid mulig å kvantifisere slikt og derfor gi kreditt for forebygging av problemer.

Målsetningen med forebygging av feil er å sikre at dersom en feil er identifisert og behandlet så vil den ikke forekomme igjen. Dette utføres ved først å analysere de viktigste årsakene til ulike feiltyper, deretter finne metoder for å hindre at disse feilene oppstår, og til slutt implementere tiltak som sikrer at disse metodene følges opp.

Implementasjon av forebygging av feil utføres i følgende trinn:

- Feilrapportering med tilstrekkelig informasjon til å klassifisere feiltyper og finne årsaken til dem.
- Årsaksanalyser foretas
- En plan må lages med tildeling av ansvar for forebygging av feil og prioritering mellom feiltyper.
- Når feilforebyggende tiltak er bestemt må de implementeres.
- Kontroller effektivitet i feilforebygging

Tidligere har fokus vært å avdekke feil og finne ut hvordan de kan rettes opp, eller kanskje også noen ganger hvordan de lettere kan oppdages. Nå er fokus rettet mot årsaken til at feilene oppstod, og hva som kan gjøres for å hindre at de oppstår igjen.

Inspeksjoner for å finne årsaker til feil, og foreslå tiltak for å forebygge dem, utføres tilsvarende formelle inspeksjoner, inkludert rapportering av data om selve arbeidsprosessen (tid til forberedelse, feiltyper behandlet, osv).

En egen arbeidsgruppe opprettes for å arbeide med forebygging av feil. Deres ansvar er å prioritere feiltyper, foreslå tiltak for å forebygge dem, sørge for at tiltakene utføres, rapportere til ledelse, osv.

En viktig bi-effekt av forebygging av feil er at alle involverte blir oppmerksomme på feiltyper, årsaksforhold og hvordan de kan forebygges.

Vedlikehold av tidligere utviklet programvare, dvs forebygging av feil etter endringer, er antagelig det mest fruktbare stedet å starte med forebygging av feil da effekten vil antagelig bli størst her. Alternativt, dersom organisasjonen allerede har data om frekvensen av feil vil det gi en god indikasjon hvor det er best å begynne.

Forebygging av feil tar tid. Dersom det ikke oppnås resultater etter 6 mnd er det antagelig noe som gjøres feil.

18. Prosessautomatisering

Fordelen med automatisering er at det generelt er mer effektivt å eliminere feil enn å utføre oppgaver mer effektivt. Det største problemet ved utvikling av store programvareprodukter er kompleksitet i programvaren og all nødvendig menneskelig interaksjon som er påkrevd. Verktøy som støtter utvikling ved kontroll med avhengigheter mellom programvarekomponenter kan derfor gi verdifull hjelp.

Kap.18 vurderer ulike sider ved det å definere en strategi og legge planer for automatisering av utviklingsprosessen, men dette er ofte veldig avhengig av de faktiske verktøyene som tas i bruk.

19. Kontrakter om Programvareutvikling

19.1 Tillit

Tillit mellom kunde og leverandør er viktig ved inngåelse av en kontrakt. Dersom kunden ikke har tillit til leverandøren vil dette typisk medføre krav om:

- Starte med presise krav om hva som er ønsket
- Insistere på rigide standarder og detaljert dokumentasjon av hver fase
- Krav om at hver fase er avsluttet før neste påbegynnes
- Krav om en utvetydig avtale i utgangspunktet (f.eks fastpris e.l)

Dette er i bunn og grunn fossefallsmodellen for programvareutvikling.

Tillit etableres først og fremst gjennom en effektiv utviklingsprosessen hvor kunden hele tiden har oversikt over status og fremdrift og derfor kan sammenholde denne med de opprinnelige planer. En fordel med et høyere modenhetsnivå er nettopp at det er mulig, også kvantitativt, å fastsette status og fremdrift.

19.2 Endringer må behandles som et naturlig element i utviklingsprosessen

Et viktig prinsipp for enhver kontrakt er å innse at endringer i kravet til et produkt vil komme under utviklingsprosessen. Det er derfor viktig å behandle dette som en naturlig del av kontrakten fremfor å betrakte dette som en feil eller abnormalt tilfelle; dvs kravspesifikasjonen er ikke ferdig før produktet er ferdig.

De funksjonelle krav som er kjent beskrives fullt ut, men det er viktig ikke å forsøke å spesifisere krav som ikke egentlig er fullt ut forstått. Design kan starte når et tilstrekkelig grunnlag for kjent funksjonalitet er definert.

Grunnleggende målsetninger for et produkt må fastsettes initsielt, og det må lages en dokumentert plan for hvordan det skal kunne foretas validering av de oppgitte krav og målsetninger; dvs hvordan vite om de oppgitte krav og målsetninger er oppfylt. Dette kan være prototyper, tester, inspeksjoner, spørreundersøkelser, demonstrasjoner, etc. Det er også en stor fordel om disse valideringene kan foretas på et så tidlig stadium som mulig.

20. Oppsummering

20.1 Lokale tilpasninger

'Managing the Software Process' er rettet mot større bedrifter. Metodene må tilpasses mindre bedrifter hvor samme person gjerne har flere roller, og hvor visse funksjoner ikke er like viktige og derfor kan f.eks løses mer uformelt.

Humphrey understreker at dersom modenhetsmodellen gir noen aktiviteter høy prioritet, mens lokale vurderinger tilsier noe annet, bør modellen forkastes da den er kun ment som en guide med retningslinjer. Det er imidlertid viktig å ha en modell i bunn, slik at dersom en modell forkastes bør den erstattes av en annen modell.

Prosessen må alltid tilpasses de lokale forhold. Det viktigste er å komme igang, foreta tilpasninger, og publisere resultater slik at andre kan lære av disse.

20.2 Konfliktskapende?

Tiltak for prosessforbedring kan i sin natur være konfliktskapende ved motstridende interesser dersom de ulike aktiviteter, særlig inspeksjoner o.l, betraktes isolert fra helheten - 'bedriftens beste'.

Humphrey vektlegger i alle handlinger og faser betydningen av å skape 'en god stemning' med belønning o.l for gode resultater oppnådd. Generell velvilje fra alle involverte er avgjørende.

Personlig modenhet er viktig; 'en for alle, alle for en'; unngå allianser som forhindrer den ønskede virkning - 'jeg hjelper deg, du hjelper meg'.

20.3 Konsekvenser av mislykket prosessforbedring

Selv om det ikke finnes resultater som utvetydig viser at prosessforbedring lønner seg, er det grunn til å tro at prosessforbedring ihvertfall lønner seg dersom det blir en suksess.

Spørsmålet man må stille m.h.p innsats versus gevinst er kostnaden versus sannsynligheten for suksess. 'Managing the Software Process' diskuterer ikke mulige konsekvenser av mislykket prosessforbedring. Endringer bør innføres gradvis slik at gevinst på et område motiverer til å fortsette med andre områder; forøvrig 'ikke satse mer enn man har råd til å tape'.