



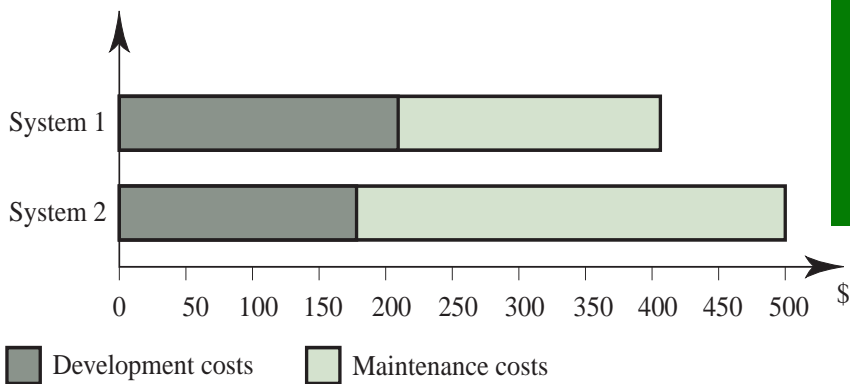
Vedlikehold og gjenbruk

Magne Jørgensen

INF3120-1



Development/maintenance costs





Definisjon

Vedlikehold av programvare omfatter alle endringer til et programvaresystem etter at det er installert for første gang i en operasjonell omgivelse

INF3120-3



Er vedlikehold nødvendig?

- Er ikke vedlikehold bare nødvendig fordi man ikke gjorde det helt riktig første gang?
- NEI, fordi: Virkeligheten endrer seg hele tiden
 - firma/organisasjonen (produksjonen, produkter, omorganisering, oppkjøp, nye konkurrenter,..)
 - brukeres behov vokser/endres
 - lover, regler og andre krav fra samfunnet
 - maskiner, operativsystem,...
- Brukerne greier aldri å spesifisere alt (de burde ha). Når man er ferdig med systemet, har spesifikasjonen endret seg (noe)
 - Et system som ikke vedlikeholdes, er etter kort tid ubrukelig

INF3120-4



Er vedlikehold annerledes enn ny-utvikling?

- ❑ Vedlikehold endrer eksisterende, ofte ganske gammel kode (dagsverk). Nyutvikling lager et nytt stort system (årsverk)
- ❑ Vedlikehold er oftest styrt av ytre hendelser, hver oppgave kan ikke planlegges som ved ny-utvikling
- ❑ Det overstående gjelder ikke like mye hvis vi nytter en evolusjonær (iterativ) systemutviklingsmodell (med gjenbruk og prototyper). Da er forskjellene mindre

INF3120-5



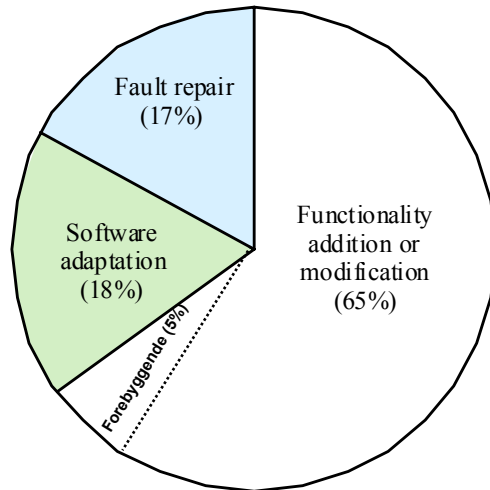
Typer vedlikehold

- ❑ **Feilrettinger (corrective maintenance)**
 - Oppdatere og rette opp feil – debugging (fault repair)
- ❑ **Tilpasninger (adaptive)**
 - Tilpasninger til endringer i omgivelsene – inkludert maskin- og systemprogramvare
- ❑ **Forbedringer (perfective)**
 - Nye krav fra brukerne, forbedret dokumentasjon, forbedret ytelse etc.
- ❑ **Forebyggelse (preventive)**
 - Forbedringer for å forhindre framtidige feil (opprydding)

INF3120-6



Distribution of maintenance effort

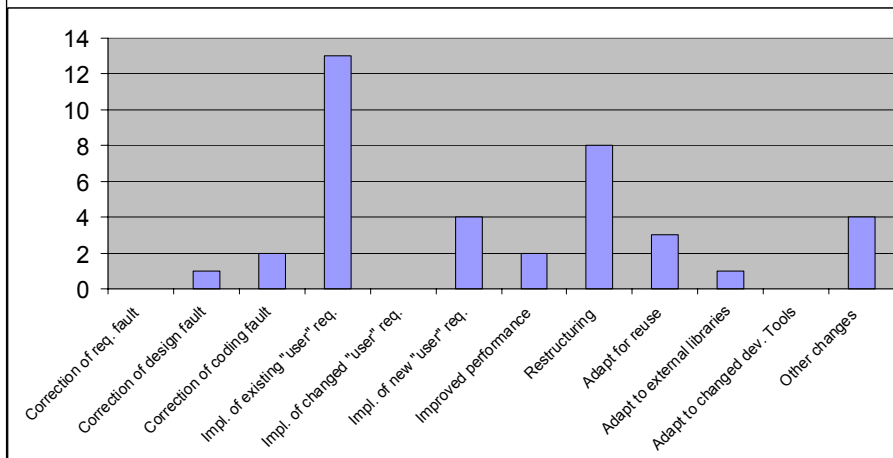


© Ian Sommerville 2000

INF3120-7



Typer endringer i ett prosjekt



Erik Arisholm 8.11.2002

INF3120-8



Vedlikeholdsprosessen

Ofte dyrere å legge til funksjonalitet senere:

- Vedlikeholderne kjenner ikke problemområdet godt
- Opprinnelig systemutvikler ikke lenger tilgjengelig
- Gammel kode og gamle kompilatorer/verktøy gjør alt vanskeligere
- Endringer utført av mange ulike personer medfører ofte uoversiktlig og kompleks kode
- Dokumentasjonen mangler eller er ikke vedlikeholdt

INF3120-9



Vedlikehold og ny-utvikling

- Vedlikehold har ofte lav status (ufortjent)
- Vedlikeholdskostnadene stiger med tiden i en organisasjon
 - Stadig nye systemer, få skrinlegges
 - Gamle systemer koster stadig mer å vedlikeholde
 - Svært mange prosjekter som består i å lage helt nye systemer som erstatning for gamle systemer, mislykkes (jfr. legacy systems)
- Vedlikehold/videreutvikling må planlegges under selve utviklingen av første versjon

INF3120-10



Foreldet dokumentasjon

- ❑ Programvare-dokumentasjon er notorisk dårlig og nesten alltid foreldet
- ❑ Den eneste oppdaterte dokumentasjonen er stort sett kildekoden selv eller dok. som er automatisk generert fra den.

INF3120-11

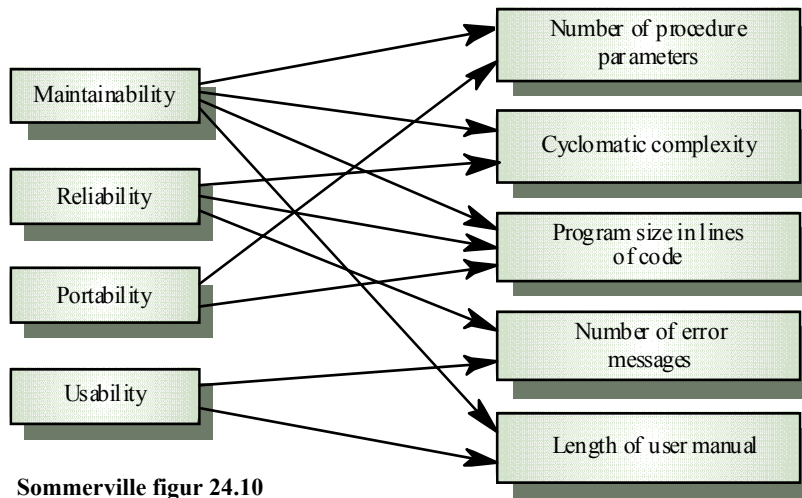


**Hvordan måle/evaluere
vedlikeholdbarheten/
endringsevnen til et system?**

INF3120-12



Eksterne og interne attributter



Sommerville figur 24.10



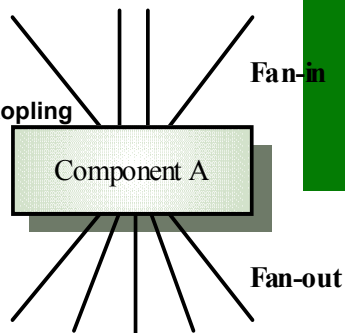
Vedlikeholdsmetrikker – produkt

- **Kontroll-kompleksitet** – kompleksiteten til betingelsesstrukturene
- **Data-kompleksitet** – kompleksiteten til data-strukturer og komponent-grensesnitt
- **Lengde på identifikatorer** – lengre navn vil kunne indikere større lesbarhet
- **Program-kommentarer** – mer kommentering vil kunne indikere enklere vedlikehold
- **Kobling** – mengden av andre komponenter/data-strukturer benyttet angir sannsynlighet for feil etter endring
- **Grad av bruker-interaksjon** – jo mer bruker-I/O, jo mer sannsynlig er det at komponenten vil kreve endring
- **Ytelses- og plasskrav** – krever innviklet programmering som indikerer vanskeligere vedlikehold



Design-metrikker

- Lav kobling og høy intern sammenheng ⇒ god vedlikeholdbarhet
- Strukturell “inn- og utvifte”
 - Stor “innvifte” (antall kallende funksjoner) indikerer høy import-kobling pga modul-avhengighet
 - Stor “utvifte” (antall funksjoner som blir kalt) indikerer høy kontroll-kompleksitet

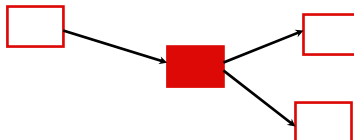


INF3120-15

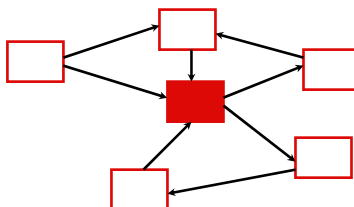


Lav vs. høy kobling mellom moduler (klasser)

V1: lav kobling



V2: høy kobling



INF3120-16



Vedlikeholdsmetrikker – prosess

- Andel korrektiv vedlikehold (feilretting)
- Gjennomsnittlig tid som kreves for konsekvensanalyse
- Gjennomsnittlig tid for å implementere en endringsforespørsel
- Antall endringsforespørsler som ikke er implementert

En økning av én eller flere av disse vil kunne indikere problemer mht. vedlikeholdbarhet

INF3120-17



Gjenbruk av programvare

INF3120-18



Kategorier av gjenbrukbare komponenter

- Totalsystemer/delsystemer
- Delsystemer
- Moduler/objekter
 - Klassebiblioteker og andre biblioteker
- Funksjoner
 - Funksjonsbiblioteker

INF3120-19



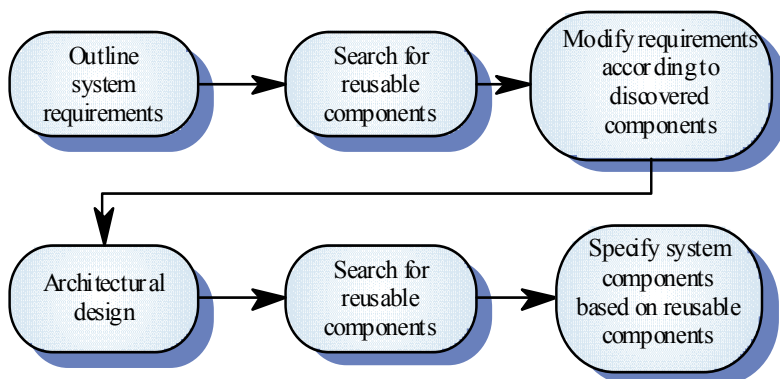
Suksessfaktorer ved gjenbruk

- Det må være mulig å gjenfinne gjenbrukbare komponenter
 - Hvordan dokumentere dem? Teknikker fra fritekstsøking (Information Retrieval), jfr. alltheweb, Alta Vista etc.
- Gjenbrukere må kunne forstå komponentene og ha tillit til at de vil møte de nødvendige behov
- Komponentene må ha dokumentasjon som sier hvordan de kan gjenbrukes
- Rettighetsproblemer må håndteres på en enkel måte
- Ansvarsforhold på være avklart
- Ved feil må det være mulig å kontakte et støtteapparat, eller ha tilgang til kildekoden

INF3120-20



Systemutvikling med gjenbruk og tilpassing av krav



Problemer ved gjenbruk

- ❑ Vanskelig å kvantifisere kostnadene og fordelene
- ❑ Mange CASE-verktøy støtter ikke gjenbruk på en god måte. De kan generelt ikke integreres med biblioteker av gjenbrukbare komponenter
- ❑ Noen programutviklere ønsker å omskrive fremfor å gjenbruke komponenter
- ❑ Teknikker for klassifisering, katalogisering og gjenfinning er umodne



Kostnader

- ❑ Dyrere å lage gjenbrukbare komponenter enn spesifikke. Denne ekstra kostnaden bør være en organisasjonskostnad fremfor en prosjektkostnad. Likevel kan gjenbruk svare seg innen ett prosjekt (2-2.5 år)
- ❑ Generelle komponenter bruker ofte mer plass og har dårligere ytelse

INF3120-23



Standarder for komponenter

- Standarder – forutsetning for effektiv bruk av komponenter i et distribuert miljø (ofte ulike kom.protokoller, op.systemer osv.)
- Eks. på objekt-orienterte distribuerte rammeverk er CORBA (Common Object Request Broker Architecture)
- Kommunikasjonen mellom komponenter i distribuerte systemer baseres i stadig større grad på Internett-teknologi
 - » Suns Enterprise JavaBeans, Microsofts Component Object Model (COM)
 - » Javas plattformuavhengige og distribuerte egenskaper gjør språket og omgivelsene egnet til å utvikle og bruke komponenter i ulike miljøer

INF3120-24



Gjenbruk av prosess-egenskaper

- ❑ **Prosessmodeller**
 - **Prosesshåndbøker og andre prosessbeskrivelser**
- ❑ **Estimering- og risikomodeller**
 - **Kostnadsmodeller, sjekklister**
- ❑ **Metrikker**
 - **Egenskaper ved programvare og -prosesser**
- ❑ **Data**
 - **Kvalitative og kvantitative data, analyse-resultater**