Konfigurasjonsstyring – Kap. 29

I. Konfigurasjonsstyring og versjonshåndtering

II. Verktøy for konfigurasjonsstyring

III. Systembygging - hvordan gjenoppbygge systemer etter endringer?

IV. Verktøy for systembygging

V. Endringshåndtering generelt

Del I: Konfigurasjonsstyring (1)

- **Konfigurasjonsstyring – disiplin for å håndtere endringer og ulike versjoner av komponenter**
- **Konfigurasjonsstyringsverktøy – støtter håndtering av versjoner av komponenter og konfigurering (sammensetninger) av dem til et system**

Konfigurasjonsstyring (2)

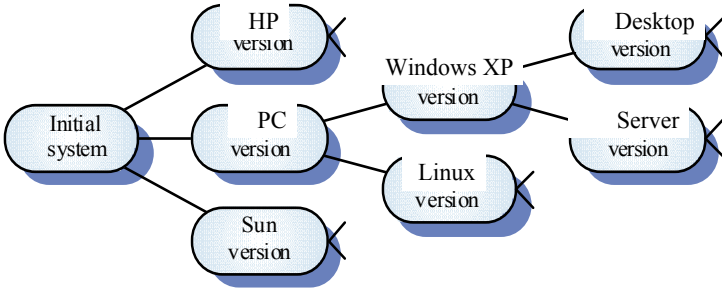
- Nye versjoner av et programsystem lages etter hvert som det endres
  - For ulike maskiner/OS
  - Tilbyr ulik funksjonalitet
  - Skreddersys for spesielle brukerkrav
- Konfigurasjonsstyring støtter evolusjon av systemer på en kostnadseffektiv og kontrollert måte
- Systemendringer er en team-aktivitet

Viktigheten av kunnskap om konfigurasjonsstyring og bygging

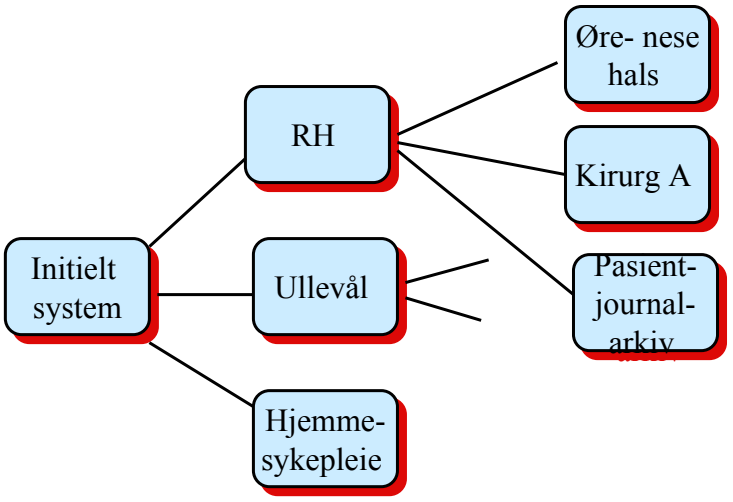
- Helt avgjørende i nærmest all industriell programvareutvikling
- Likevel finnes nesten ingen kompetanse om dette hos nyutdannete kandidater fra universitet/høyskole

Konfigurasjon

Konfigurasjon – samling av alle komponentene til et system hvor hver komponent er representert med nøyaktig én versjon som er valgt i hht et bestemt kriterium, f. eks. siste versjon, plattform, spesiell funksjonalitet



Vaktbyttesystem



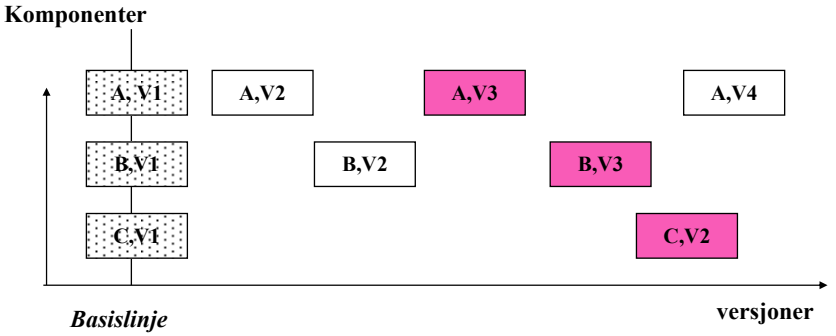
Konfigurasjonsstyring for mange typer systemer

- Programvare
- Maskinvare
- Ingeniørprodukter
- Bøker

Konfigurasjonsstyring kan omfatte alle typer systemprodukter

- Spesifikasjoner
  - Tekstlige kravspesifikasjoner, Use Case diagrammer etc.
- Design-dokumenter
  - UML klassediagrammer
  - Databaseskjema
  - Skjermbilder
- Programmer
  - Javakode
  - Scripts
  - Lagrede prosedyrer i databaser
- Test-data
- Brukermanualer

Versjoner og konfigurasjoner



- En komponent kan foreligge i flere *versjoner*.
- Et fullstendig sett med komponenter i bestemte versjoner utgjør en *konfigurasjon*.

Basislinjer

- **Basislinje:**

  Spesiell, kontrollert konfigurasjon som fungerer som plattform for videreutvikling av systemet

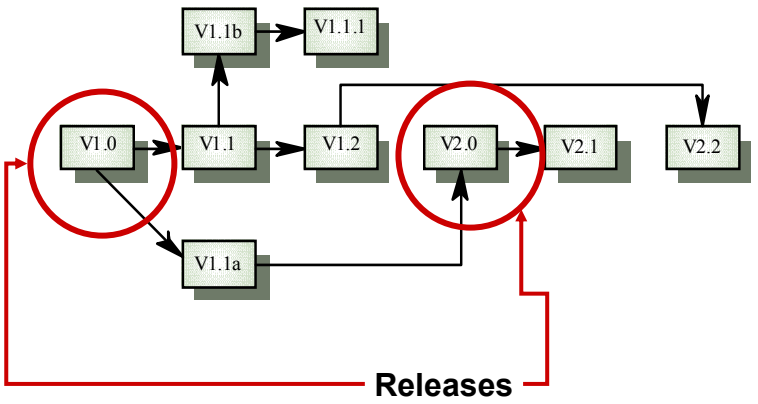
- **Vanligvis finnes det basislinjer for**
  - **Utvikling**
  - **Systemtest**
  - **Produksjon**
- **Finnes graverende eksempler på utviklingsfirmaer der man bare har hatt en basislinje – produksjon!**

Versjoner/varianter/releaser av systemer

- *Versjon* – en instans av et system som er funksjonelt forskjellig fra andre system-instanser
- *Variant* – en instans av et system som er funksjonelt identisk, men forskjellig fra andre system-instanser mht feil, ytelse etc.
- *Release* – en instans av et system som distribueres til brukere utenom utviklingsteamet

Alle disse instansene utgjør en konfigurasjon

Version/release-struktur



Versjon/release-håndtering

- Definer notasjon for identifisering av system-versjoner
- Definer kriteriene for når en ny versjon eller ny release skal genereres (ikke trivielt)
  - Kunder ønsker ikke alltid en ny release av et system: De kan være fornøyd med eksisterende versjon og ønsker ikke ny funksjonalitet (f.eks. MS Word)
  - Ikke anta at kundene har installert alle tidligere releaser. Alle filer som kreves for en release må genereres på nytt når releasen skal installeres
- Sjekk at rutiner og verktøy for versjonshåndtering anvendes som planlagt
- Planlegg og distribuer nye system-releaser

Håndtering av komponenter/dokumenter

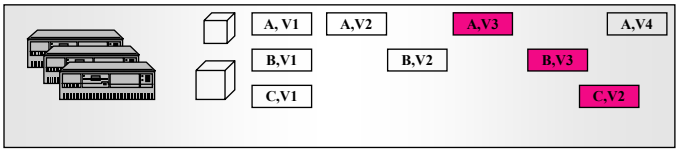
I store programsystemer kan flere tusen dokumenter genereres og potensielt lagres lenge

- Må kunne identifiseres (navnekonvensjoner),
  - V1.0, V2.0, V3.0 for releaser, etc.
- Standardisering påkrevet
  - Konfigurasjonsstyring bør baseres på en mengde standarder i en organisasjon
  - Standardene bør definere hvordan komponenter identifiseres, hvordan endringer kontrolleres og hvordan nye versjoner håndteres
  - Finnes int. standarder som kan brukes som utgangspunkt (IEEE, ISO etc.), men er gjerne basert på fossefallsmodellen. Nye standarder trengs for evolusjonær, inkrementell utvikling

Konfigurasjonsstyringsplan

- Definerer typer av dokumenter og navngivingskonvensjoner
- Definerer personansvar
- Beskriver aktuelle verktøy og deres begrensninger
- Definerer konfigurasjonsdatabasen som bør lagre all relevant informasjon om konfigurasjons-styringen

Konfigurasjons-databasen

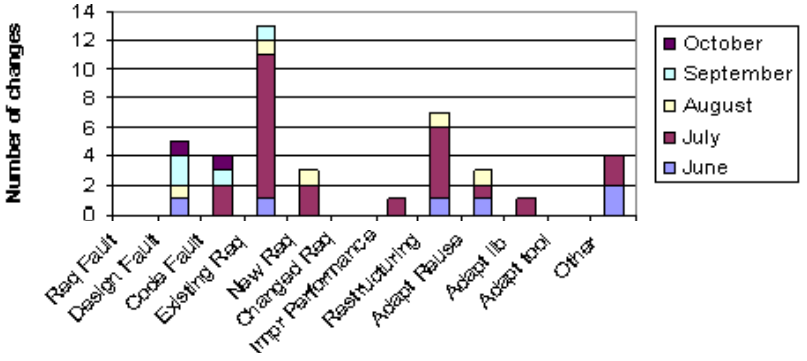


- Konfigurasjons-databasen er sentral!
- Inneholder:
  - Kildekomponenter i ulike versjoner
  - Applikasjonskonfigurasjoner: Sammenstilling av versjoner av kildekomponenter
  - Endringspakker: Sammenstilling av endringer som skal anvendes som en enhet
- Inneholder *avhengigheter* mellom objekter

Konfigurasjonsdatabasen bør kunne svar på:

- Hvem arbeider med/er ansvarlig for en bestemt systemversjon?
- Hvilken plattform kreves for en bestemt versjon?
- Hvilke versjoner påvirkes av en endring til komponent X?
- Hvor mange feil er rapportert i versjon Y av komponent Z?
- Hvor er de kritiske komponentene?
  - Hvor er det gjennomført fleste endringer?
  - Hvor er det rapportert fleste feil?
  - Hva slags type endringer gjennomføres og hva er konsekvensene?

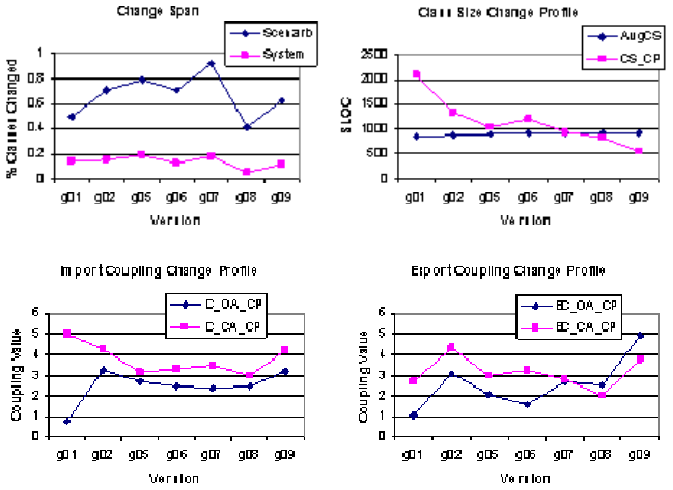
Kategorier av endringer



[Erik Arisholm, Empirical Asessement of Changeability in Object-Oriented Software, PhD Thesis, ISSN 1501-7710, No. 143, Unipub forlag, Oslo, February 2001]

- **Basert på informasjon i ClearCase (se siden)**

Utvikling av kvalitet over tid



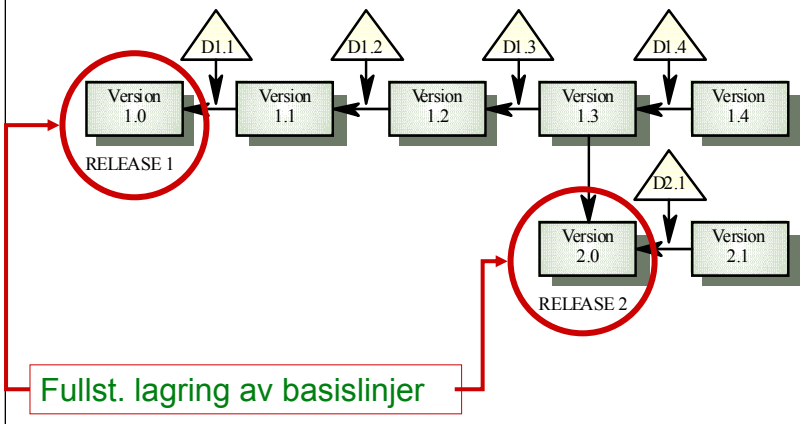
[Erik Arisholm, Empirical Assessement of Changeability in Object-Oriented Software, PhD Thesis, ISSN 1501-7710, No. 143, Unipub forlag, Oslo, February 2001]

Del II: Verktøy for versjonskontroll/ konfigurasjonsstyring

Basisverktøyene:

- SCCS (1972), RCS (1985) – håndterer multiple versjoner av tekstfiler (vanligvis kildekode-filer) i henhold til en "sjekk-ut/sjekk-in (commit)"-protokoll.
  - Verktøysystemene genererer automatisk en ny identifikator når en ny versjon legges inn i systemet
  - Lagrer motivet for hver ny versjon (hvis utvikler har lagt det inn)
- Av hensyn til diskplass lagres bare forskjellene (delta) mellom versjoner. SCCS anvender forover-deltaer (lagrer original); RCS bakover-deltaer (lagrer siste)
- RCS tillater uavhengig utvikling av forskjellige releaser
- CVS og CVS-baserte verktøty har overtatt etter RCS og SCCS

Lagring av deltaer



CVS – Concurrent Versions System

- **Overbygning til RCS - utvider versjonskontroll fra en samling filer innen et directory til en hierarkisk samling av directories som består av versjonskontrollerte filer**
- **Kontrollerer samtidige endringer av kildefiler blant mange utviklere**
- **Kan lagre binærfiler, men disse lagres fullt ut, dvs. ingen deltaer som beskriver forskjellen mellom versjoner. Videre er det ingen automatisk håndtering av avhengigheter mellom kilde-komponenter og genererte binær(objekt)-filer**

Bruk av CVS

Framgangsmåte for å lage ny versjon:

1. `cvs checkout`

   Kan nå editere på en lokal kopi av dok. (filen) du ønsker å lage en ny versjon av

2. `cvs diff`

   Før du "sjekker inn" bør du vite hva du faktisk har andret. Du får oversikt over dette, ved å utføre cvs diff

3. `cvs commit`

   Lagrer endringer i en ny versjon (heter i mange systemer "check in"). Før CVS legger endringene inn i databasen, startes opp en editor og du blir bedt om å skrive inn en loggmelding.

3. `cvs update`

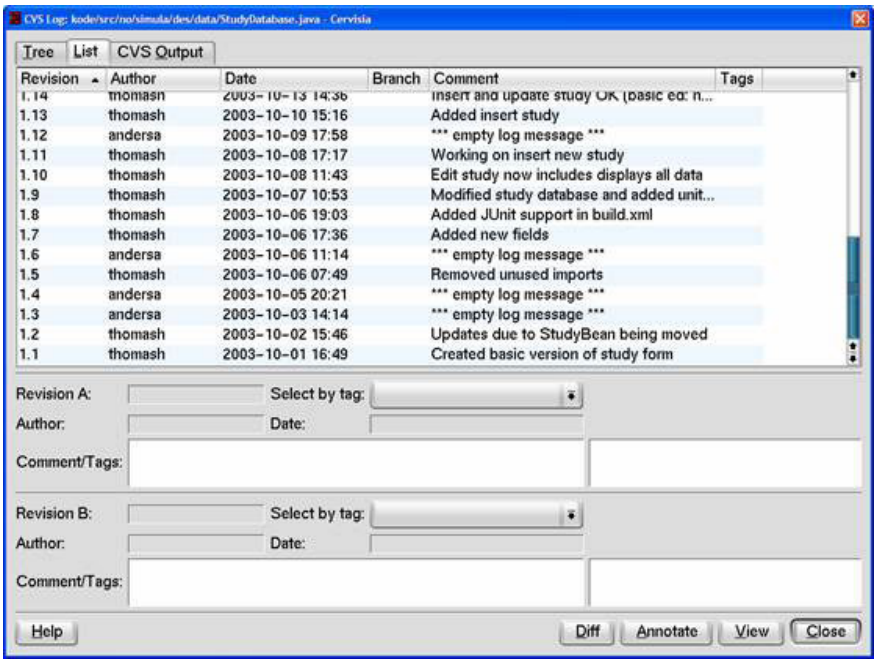
   Hvis flere jobber i samme prosjekt som potensielt kan endre de samme dokumentene, bruk cvs update før du sjekker inn (commit). Du vil da få din arbeidskopi oppdatert (endret) med de endringene som er foretatt og "committed" av andre på den samme versjonen som du "sjekket ut" fra. Evt. konflikter må håndteres manuelt.

Se også Unix man page og "lokal guide til CVS" (norsk) på INF3120 web

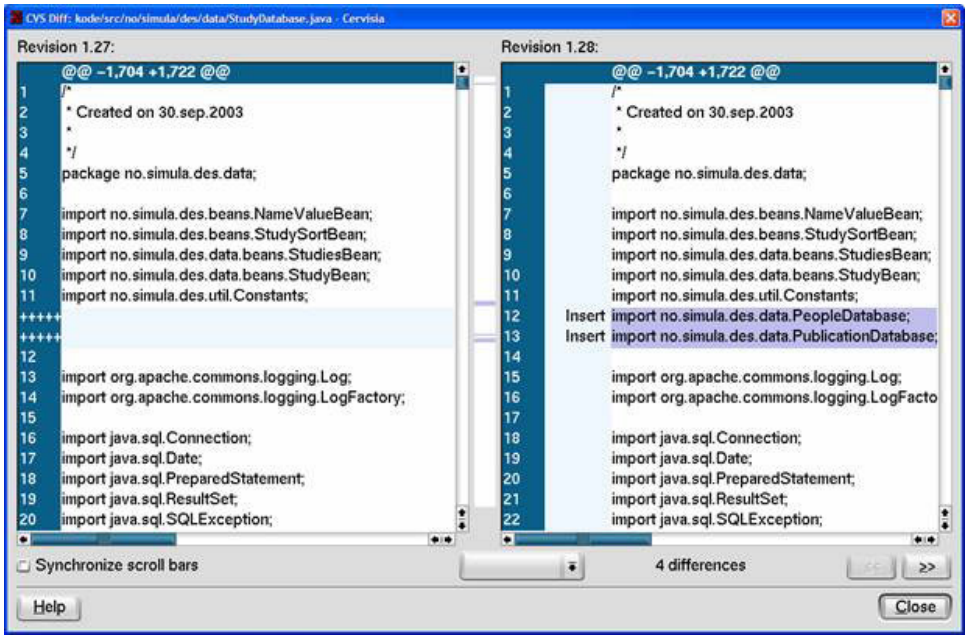
Open-source grafisk grensesnitt mot CVS

- **Cervisia: http://www.kde.org/apps/cervisia/overview.html**
  - **Updating or retrieving the status of a working directory or single files. Files are displayed in different colours depending on their status, and the shown files can be filtered according to their status**
  - **Common operations like adding, removing and committing files.**
  - **Advanced operations like adding and removing watches, editing and unediting files, locking and unlocking.**
  - **Checking out and importing modules.**
  - **Graphical diff against the repository and between different revisions.**
  - **Blame-annotated view of a file.**
  - **View of the log messages in tree and list form.**
  - **Resolving of conflicts in a file.**
  - **Tagging and branching.**
  - **Updating to a tag, branch or date.**
  - **A Changelog editor coupled with the commit dialog.**
- **ViewCVS: http://viewcvs.sourceforge.net/**
  - **ViewCVS can browse directories, change logs, and specific revisions of files. It can display diffs between versions and show selections of files based on tags or branches. In addition, ViewCVS has "annotation" or "blame" support, Bonsai-like query facilities, template-based page generation, and support for individually configuring virtual hosts. It also includes support for *CvsGraph* -- a program to display the tree of revisions and branches graphically.**
  - **Currently, the functionality of ViewCVS surpasses that of cvsweb.**

Historikken til en fil



Forskjell mellom versjoner



Systemprodukter (objekter) i Tau UML som har versjoner

- **When you create, delete, or modify these types of objects, you always create, delete, or modify a version of the object.**
- **Eks. på objekter**
  - **Configurations**
  - **Phases**
  - **Packages**
  - **System files, such as diagrams and CDMs (class definitions)**
  - **Groups**
  - **Text files**
  - **Customization files (profiler)**

Tilstander til en versjon av et objekt i Tau UML

State	Description
working	An object version in this state can be edited. Its component object versions can be working or frozen. A working object version must be in exactly one configuration version.
frozen	An object version in this state cannot be edited. Its component object versions are also frozen. A frozen object version can be in one or more configuration versions.
backGround	An object version in this state is frozen and is not in any configuration version. Background versions are removed from the repository and stored in the file system.

Basisversjon tilsv. *frozen* object i Tau UML:

”Typically, you freeze an object version to make it available to other users or so that you can return to it. For example, you are about to change the model, but you want to be able to return to the current version if the changes do not work.”

Lage ny versjon i Tau UML

- Utfør kommandoen **`Version|Freeze`** på det nivået du ønsker (fase/pakke/diagram/fil). Tau traverserer da rekursivt ned til bunnen av treet og fryser alt som ligger der.
- Lage en ny versjon: Gå til den aktuelle pakken, marker så den el. de diagrammer/filer som du ønsker å lage en ny versjon av og utfør **`Version|New.`** Tau traverserer da treet oppover og lager nye versjoner av alt som ble frosset ovenfor. Du kan nå jobbe på den nye versjonen.
- Ønsker du å jobbe videre på en ny konfigurasjon (configuration), gjør da følgende: Frys det du vil jobbe videre med. I den nye konfigurasjonen utfør **`Version|Select|New.`** Du får da opp en liste med dem du kan velge fra. Gjør valget og lag en ny versjon av det du nettopp hentet inn i den nye konfigurasjonen som beskrevet ovenfor.
- Se også Tau tutorial, manual på Web

Versjonsnummerering i Tau UML

Object Version Being Created	Rule For Determining New Version Number	Example (Frozen Version -> New Version)
First object version	Version is 1.	1
First object version based on a frozen version	Increment the last part of the version number.	1 -> 2 or 1.1 -> 1.2
Second object version based on the same frozen version	Add .1 to the frozen version number.	1 -> 1.1 or 1.1 -> 1.1.1
Third object version based on the same frozen version	Add .0.1 to the frozen version number.	1 -> 1.0.1 or 2.2 -> 2.2.0.1

Generasjoner av konfigurasjonsstyringsverktøy

- **1. generasjon: SCCS, RCS, CVS,**
- **2. generasjon:**
  - **DSEE (forløper til ClearCase)**
  - **Microsoft® Visual SourceSafe™ (msdn.microsoft.com/ssafe/default.asp)**
  - **Subversion http://subversion.tigris.org/ (har versjonskontroll av kataloger, samt filnavn-endringer) (se linker på INF3120-siden, er installert på Ifi)**
  - **PVCS www.pvcs.synergex.com**
  - **Perforce www.perforce.com/**
  - **Tau UML (ikke generelt, men for objekter som lages i Tau)**
  - **Web-baserte overbygninger til CVS**

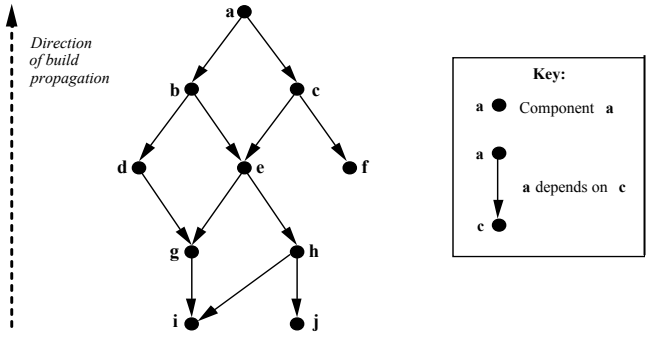
  (Det finnes et utall gratisverktøy og kommersielle verktøy i denne klassen, i tillegg til at hver organisasjon ser ut til å implementere sitt eget… IKKE GJØR DET!)
- **3. generasjon: ClearCase www-306.ibm.com/software/awdtools/clearcase**

I motsetning til gjenoppbyggingsavhengigheter (se senere), må identifisering av gyldige konfigurasjoner fortsatt gjøres manuelt.

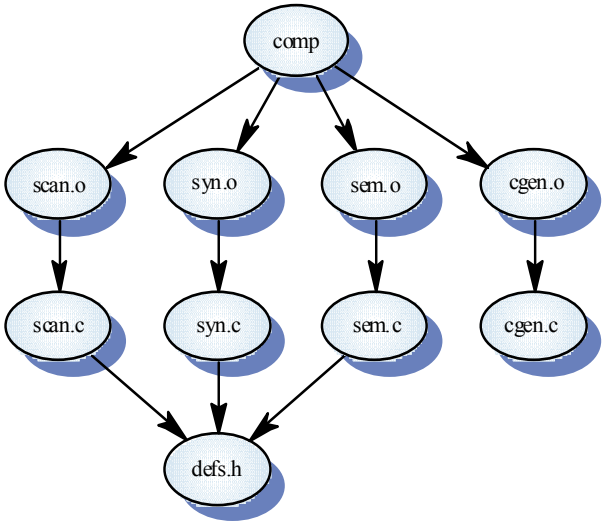
Del III: Systembygging (build management)

- **Store programsystemer består av mange enkelt-komponenter som sammen utgjør en avhengighetsgraf. Når en kilde-komponent endres, må avledede komponenter (gjenopp)bygges.**
- **Systembygging kontrollerer bygging av avledede komponenter som utgjør det komplette kjørbare systemet, f.eks. eksekverbar objekt-kode produsert av en kompilator ut fra kildekode-komponenter**
- **Systembygging eller gjenoppbygging involverer kompilering og linking.**
- **I avanserte miljøer omfatter avledede komponenter også kryssreferanse-databaser og annen informasjon automatisk generert fra kildekoden (dokumentasjon, database-indekser etc.)**

Generell avhengighetsgraf



Avhengighetsgraf



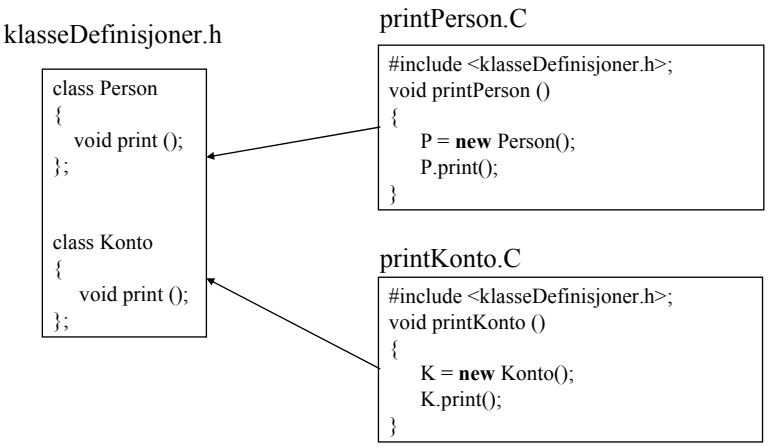
Problemer i systembygging

- Omfatter byggingsinstruksjonene alle nødvendige komponenter?
  - Lett å glemme noen når hundrevis av komponenter utgjør totalsystemet. Dette bør oppdages av linkeren, evt. av analyseverktøy
- Er den riktige versjonen av komponentene spesifisert?
  - Et system bygd med gale versjoner kan fungere i starten men feile etter levering hos kunden
- Er alle datafiler tilgjengelige?
  - Byggingsprosessen bør ikke basere seg på at "standard-filer" (f.eks. fontdefinisjoner) er tilgjengelige. Standarder varierer fra sted til sted
  - Er systemet bygd for den riktige plattformen?
  - Spesielle krav for OS-versjon el. maskinkonfigurasjon?
- Er den riktige versjonen av kompilatoren og andre verktøy spesifisert?
  - Ulike versjoner av en kompilator kan generere forskjellig objekt-kode som vil kunne oppføre seg forskjellig

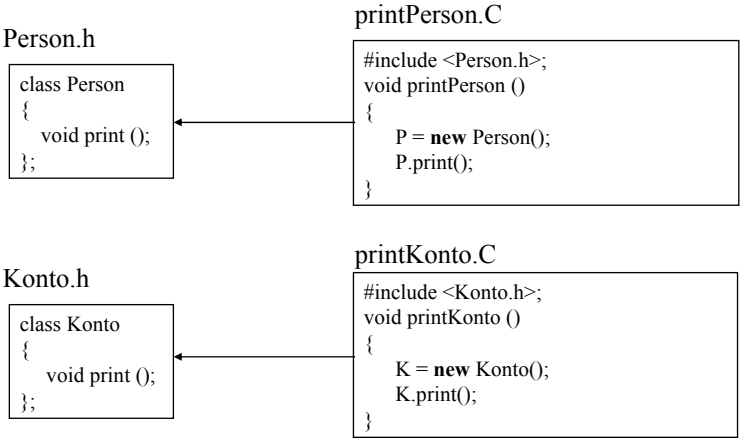
Del IV: Verktøy for bygging *(Make)*

- *Make* er det klassiske støtteverktøyet for Unix (finnes også på PC); tilsvarende verktøy for andre systemer
- Avhengigheter på fil-nivå, dvs grovkornet
- Brukes hovedsakelig til kompilering og linking, men kan initiere vilkårlige operasjoner
- Avhengigheter må utledes manuelt (visse typer avhengigheter kan avledes automatisk) og beskrives i *Makefiler. Make* starter kompilering når den oppdager at kildekode-filer er endret etter at objektkode-filer ble lagd. Unngår altså å rekompilere alt
- Programmererne må forsikre seg om at de refererte filer eksisterer
- Å skrive og håndtere (kanskje hundrevis av) *Makefiler* manuelt kan være en håpløs oppgave

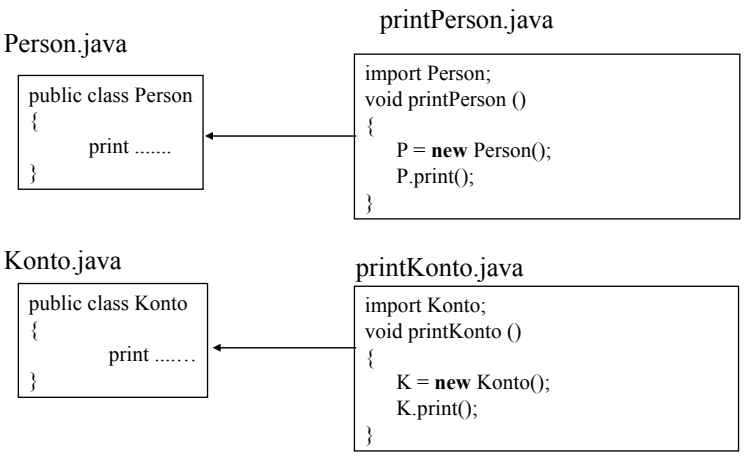
Avhengigheter – C++



Avhengigheter – C++



Avhengigheter – Java



Filene før bygging:

```
-rw-------   1 dagsj      147 Sep 27 14:56 printKonto.C
-rw-------   1 dagsj      474 Sep 27 14:45 Konto.h
-rw-------   1 dagsj      3376 Sep 27 14:46 printKonto.o
-rw-------   1 dagsj      123 Sep 27 14:12 printPerson.C
-rw-------   1 dagsj      355 Sep 27 13:48 Person.h
-rw-------   1 dagsj      1764 Sep 27 14:45 printPerson.o
-rwx------   1 dagsj      447256 Sep 27 14:46 app*
-rw-------   1 dagsj      302 Sep 27 14:59 main.C
-rw-------   1 dagsj       10524 Sep 27 14:45 main.o


beyla: test>Make
---------- Depending printKonto.C ----------

-------------- printKonto.C ---------------
CC  -DSPARC_Cplusplus -DARRAY_RANGECHECK -DHANDLE_0PTR_CHECK -DXMDPMENUS -DNUMT=double
         -I. -I/local/x11/include -I/local/share/solaris/Workshop-
         4.0/SUNWspro/SC4.2/include -I/local/share/solaris/Workshop-
         4.0/SUNWspro/SC4.2/include/CC
         -I/home/grotte/c/stpvim/bt/src/include  -o printKonto.o -c printKonto.C

---------- linking ./printKonto.o ./printPerson.o ./main.o ----------

CC  -L. -L/local/share/solaris/Workshop-4.0/SUNWspro/SC4.2/lib -L/usr/local/X11R5/lib
         -L/usr/ccs/lib -R/usr/local/X11R5/lib -R/usr/ucblib
         -L/usr/openwin/lib -R/usr/openwin/lib
         -L/home/grotte/c/stpvim/bt/src/lib/sol/nopt
         -o app ./printKonto.o ./printPerson.o ./main.o -lbt2 -larr1 -lbt1  -lF77
         -lm -lXm -lXt -lXext -lX11 -lgen -lsocket -lnsl

---------- application "app" done ----------

#Dermed blir printKonto.C kompilert til printKonto.o og alt sammen linkes til app.
```

Makefile eks.

```
#Utsnitt fra MakeHeader
# C++ compiler:
CXX         := CC


#Utsnitt fra MakeRules:
$(PREFIX)%.o: %.C
        $(CXX) $(CXXFLAGS) -o $@ -c $<



#Utsnitt fra MakeFlags:
ifeq ($(HOSTTYPE),sol)
    SPARC_DEFINES  = -DSYSV -DSVR4 -DSTRINGS_ALIGNED \
                     -DNO_REGCOMP -DINCLUDE_ALLOCA_H

    CCOPTIONS := $(CCOPTIONS) $(SPARC_DEFINES) \
                    -I/usr/local/X11R5/include

    LIBS := $(LIBS) -lXm -lXt -lXext -lX11 -lgen -lsocket

endif


#Utsnitt
$(APPL): $(OBJS) $(DEPLIBS)
        @echo "---------- linking $(OBJS) ----------"
        $(LD) $(LDFLAGS) -o $@ $(ALLOBJS) $(LIBS) $(SYSLIBS)
```


Automatisk generering av Makefiler

- Avhengigheter mellom filer typisk uttrykt gjennom `import` (Java) og `#include` statements (C, Pascal og FORTRAN)
- Basert på enkel gjenkjenning av nøkkelord (f.eks. makrospråket til C) har flere verktøy som genererer Makefiler, blitt utviklet, f.eks. Imake, GNU Make (*makedepend*)
- Slike verktøy kan også støtte portabilitet; de besitter informasjon om kompilator-opsjoner, alternative navn på kommandoer etc. som også spesifiseres i Makefiler

Problemer ved *Make*

- Avhengighetsspesifikasjoner (Makefiler) blir fort store, komplekse (ofte helt uforståelige) og vanskelig å vedlikeholde
- Make har en primitiv modell av endringer basert på tidspunkt for endrede filer ("timestamps"). Kildekode vil ikke nødvendigvis trenge rekompilering. Verre er det at kompilering kan være nødv. selv om kildekode ikke er endret, f.eks. ved installering på en annen plattform
- Make er sjelden direkte knyttet opp mot versjonskontroll-verktøy
- Make har mange særheter, f. eks. skjelnes det mellom tab og blanke

Detaljeringsnivå på komponenter

- Grovkornet inndeling: komponenter er filer/virtuelle filer (ingen forståelse av innholdet), f. eks. *Make*
- Finkornet inndeling: komponenter er klasse/type-definisjoner, prosedyrer, konstanter, variable og andre konstruksjoner som kan uttrykkes i et programmeringsspråk.
  - Verktøy med finkornet forståelse er typisk språk-avhengige og vil kunne automatisere flere oppgaver enn språk-uavhengige systemer, f.eks. avledning av avhengighetsgrafer

Grovkornede, språk-uavhengige verktøy, f. eks. ClearCase

- Avhengigheter mellom kilde-komponentene og de avledede komponentene beskrives av brukeren i en system-modell
- Håndterer og lagrer også binærfiler som objekt-kode, bitmap grafikk etc. (utviklerne konsentrerer seg bare om kildekomponenter – i motsetning til i Make)
- Dokumentasjon av byggingsprosessene – hvilken kommando ble utført når, og hva ble produsert?

ClearCase – systembygging i nettverk

- Parallell bygging av forskjellige systemversjoner
- Parallell bygging på forskjellige noder i et nettverk (effektivt)
- Integrert versjonshåndtering, kildekodekontroll og systembygging
- Ved forespørsel om kompilering sjekkes først at det ikke allerede finnes en tilhørende objekt-fil

System-modellering

Finnes prototyper på system-modelleringsspråk (mer høy-nivå en Make) som:

- **Beskriver mapping mellom fysiske filer og logiske komponenter**
- **Definerer grensesnitt til komponenter**
- **Relasjoner mellom komponenter, f.eks. "required" og "implemented as"**

Systemmodellen analyseres og

- **En Makefile genereres**
- **De nødvendige versjonene av komponentene identifiseres via deres attributter og sjekkes ut automatisk fra for eksempel CVS**
- ***Make* kalles for å bygge systemet**

Finkornede, språk-avhengige verktøy

- **Mange open source og kommersielle systembyggingsverktøy for Java, C/C++, COBOL etc. Er typisk en del av integrerte utviklingsomgivelser (IDE) for et gitt språk (jfr. tirsdagens forelesning) og inneholdes ofte i SCM verktøy (se tidligere)**
- **Avhengigheter avledes automatisk, endringer registreres og nødvendig re-kompilering og linking startes automatisk**
- **Altså, finnes verktøy som automatisk identifiserer "build"-avhengigheter. Dette i motsetning til konfig.styring hvor det ikke finnes verktøy som kan identifisere riktige versjoner av komponenter i en konfigurasjon**

Apache Ant (http://ant.apache.org/)

- Apache Ant is a Java-based build tool. In theory, it is kind of like Make, but without Make's wrinkles.
- Why another build tool when there is already *make*, *gnumake*, *nmake*, *jam*, and others? Because all those tools have limitations that Ant's original author couldn't live with when developing software across multiple platforms. Make-like tools are inherently shell-based -- they evaluate a set of dependencies, then execute commands not unlike what you would issue in a shell. This means that you can easily extend these tools by using or writing any program for the OS that you are working on. However, this also means that you limit yourself to the OS, or at least the OS type such as Unix, that you are working on.
- Makefiles are inherently evil as well. Anybody who has worked on them for any time has run into the dreaded tab problem. "Is my command not executing because I have a space in front of my tab!!!" said the original author of Ant way too many times. Tools like Jam took care of this to a great degree, but still have yet another format to use and remember.
- Ant is different. Instead of a model where it is extended with shell-based commands, Ant is extended using Java classes. Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular Task interface.
- Granted, this removes some of the expressive power that is inherent by being able to construct a shell command such as `find . -name foo -exec rm {}`, but it gives you the ability to be cross platform -- to work anywhere and everywhere. And hey, if you really need to execute a shell command, Ant has an <exec> task that allows different commands to be executed based on the OS that it is executing on.

Hvor ofte bygge?

- Bygging kan ta lang tid for store systemer hvis alt gjøres under ett, alternativt: separat kompilering og inkrementell linking
- Daily build? http://c2.com/cgi/wiki?DailyBuild
- Kontinuerlig bygging? http://c2.com/cgi/wiki?ContinuousIntegration

Del V: Endringshåndtering generelt

Endringshåndtering omfatter mer enn konfigurassjonsstyring og bygging:

Programvaresystemer er kontinuerlig gjenstand for krav om endringer fra

- **Brukere**
- **Utviklere**
- **Markedet**
- **Myndigheter**

Oppfølging og arkivering

- **Hvordan ha oversikt over endringsstatus?**
- **Finnes verktøy som holder rede på status og sikrer at de riktige forespørslene er sendt til de riktige personene til riktig tid, ofte integrert med e-post**
- **Logging/arkivering**
  - **Hva er endret på et gitt dokument eller kode-komponent?**
  - **Hvorfor, når og av hvem ble endringen utført?**
  - **Hvis spesielle konvensjoner for kommentering i kode følges, kan slik info ekstraheres automatisk (f.eks. JavaDoc, se http://java.sun.com/j2se/javadoc/)**

Skjema for endringskrav

- **Registrerer ønsket endring, forslagsstiller, begrunnelse for endring og tidshorisont (info fra forslagsstiller)**
- **Registrerer vurdering av forslaget, konsekvens-analyse, kostnader og anbefaling (info fra systemansvarlige)**

Change Request Form

Project: Proteus/PCL-Tools **Number:** 23/94
Change requester: I. Sommerville **Date:** 1/12/94
Requested change: When a component is selected from the structure, display the name of the file where it is stored.

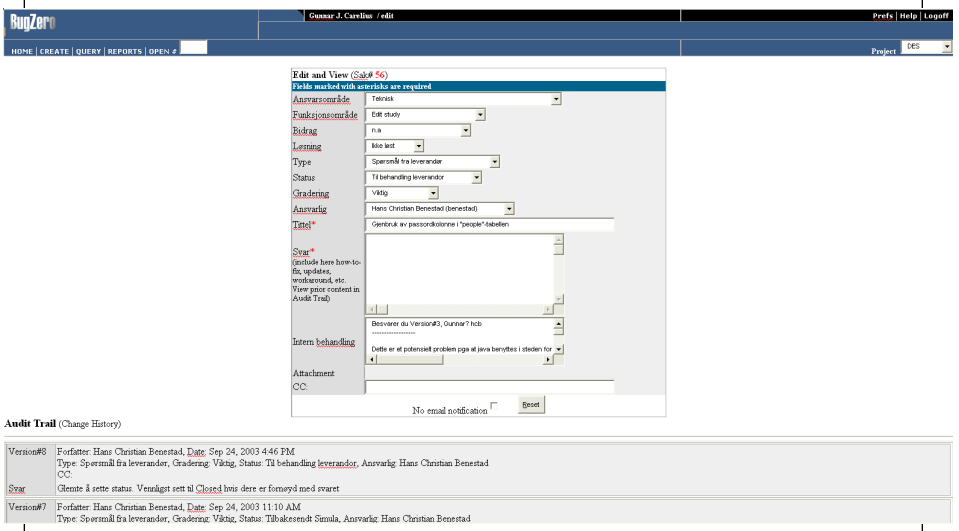
Change analyser: G. Dean **Analysis date:** 10/12/94
Components affected: Display-Icon.Select, Display-Icon.Display

Associated components: FileTable

Change assessment: Relatively simple to implement as a file name table is available. Requires the design and implementation of a display field. No changes to associated components are required.

Change priority: Low
Change implementation:
Estimated effort: 0.5 days
Date to CCB: 15/12/94 **CCB decision date:** 1/2/95
CCB decision: Accept change. Change to be implemented in Release 2.1.
Change implementor: **Date of change:**
Date submitted to QA: **QA decision:**
Date submitted to CM:
Comments

BugZero
(http://www.websina.com/bugzero/)



Oppsummering

- **Konfigurasjonsstyring er Identifisering og håndtering av komponenter som til sammen gir en versjon av et produkt (en konfigurasjon), og endringer av komponentene. Absolutt påkrevet i middels og store prosjekter**
- **Konfigurasjonsstyringsaktiviteter omfatter planlegging, formalisering av forespørsler, gjennomføring og sporbarhet av endringer, versjons- og release-håndtering og systembygging.**
- **Et navngivingsskjema for dokumenter bør etableres og de bør lagres i en database**
- **Systembygging involverer å generere og samle alle komponenter som er nødvendig for å kunne kjøre et system.**
- **Integrerte konfigurasjonsstyringsverktøy som ClearCase og Tau UML støtter både versjonshåndtering og systembygging**