Konfigurasjonsstyring – Kap. 29

I. Konfigurasjonsstyring og versjonshåndtering
II. Verktøy for konfigurasjonsstyring
III. Informasjon i konfigurasjonsdatabasen
IV. Systembygging – hvordan gjenoppbygge systemer etter endringer?
V. Verktøy for systembygging
VI. 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)

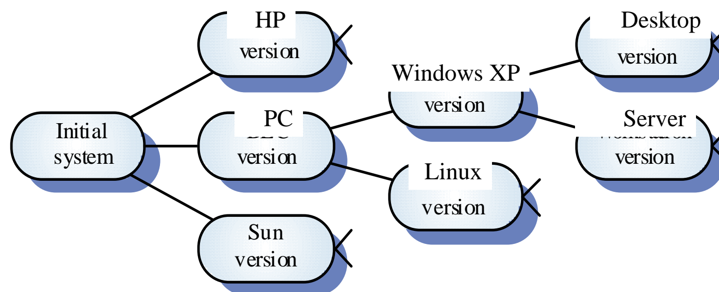
- Nye versjoner av et programsystem lages etter hvert som det endres
  - For ulike maskiner/operativsystemer
  - Tilbyr ulik funksjonalitet
  - Skreddersys for spesielle brukerkrav
- Konfigurasjonsstyring støtter evolusjon av systemer på en kostnadseffektiv og kontrollert måte
- Systemendringer er en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

Viktigheten av kunnskap om konfigurasjonsstyring og bygging

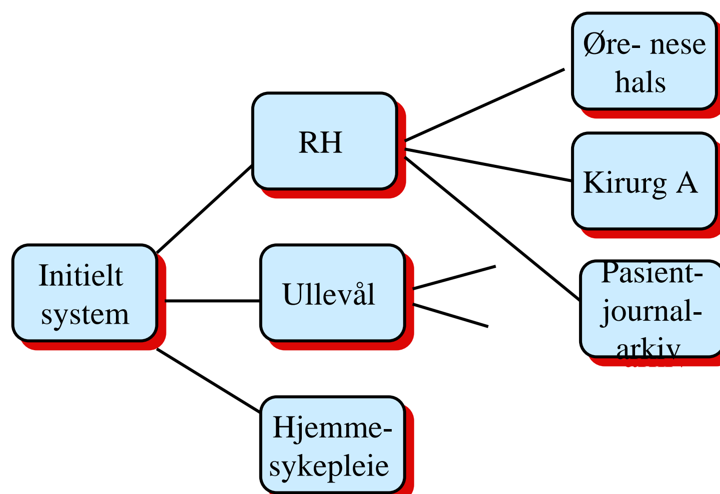
- Helt avgjørende i nærmest all industriell programvareutvikling
- Likevel finnes nesten ingen kompetanse om dette hos nyutdannete kandidater fra universitet/høyskole

Konfigurasjon

Konfigurasjon: en 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



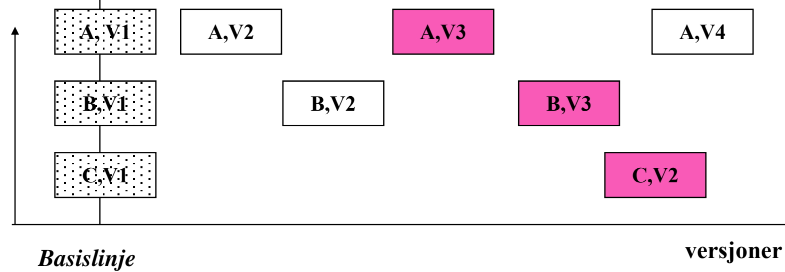
Konfigurasjonsstyring for mange typer systemer

- 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

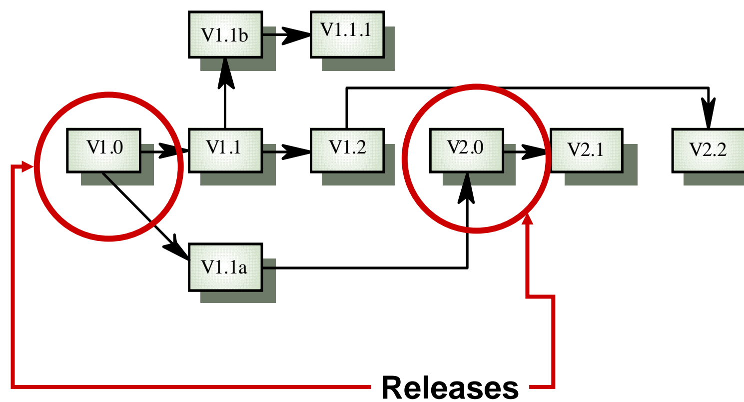
- **Vanligvis finnes det basislinjer for**
  - Utvikling
  - Systemtest
  - Produksjon
- **Finnes eksempler på utviklingsfirmaer som bare har hatt én basislinje – produksjon!**

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 systemversjoner
- 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

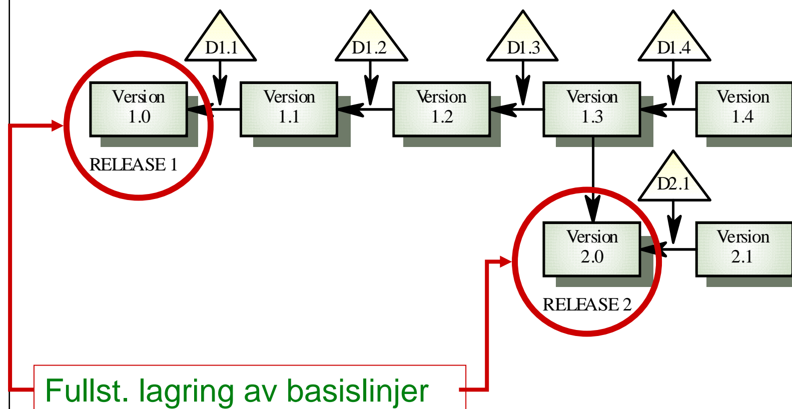
- Må kunne identifiseres (navnekonvensjoner),
  - V1.0, V2.0, V3.0 for releaser, etc.
- Standardisering påkrevet
  - Konfigurasjonsstyring bør baseres på en mengde standarder i en organisasjon
  - Standardene bør definere hvordan komponenter identifiseres, hvordan endringer kontrolleres og hvordan nye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

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 notat om CVS på INF3120 web

Subversion

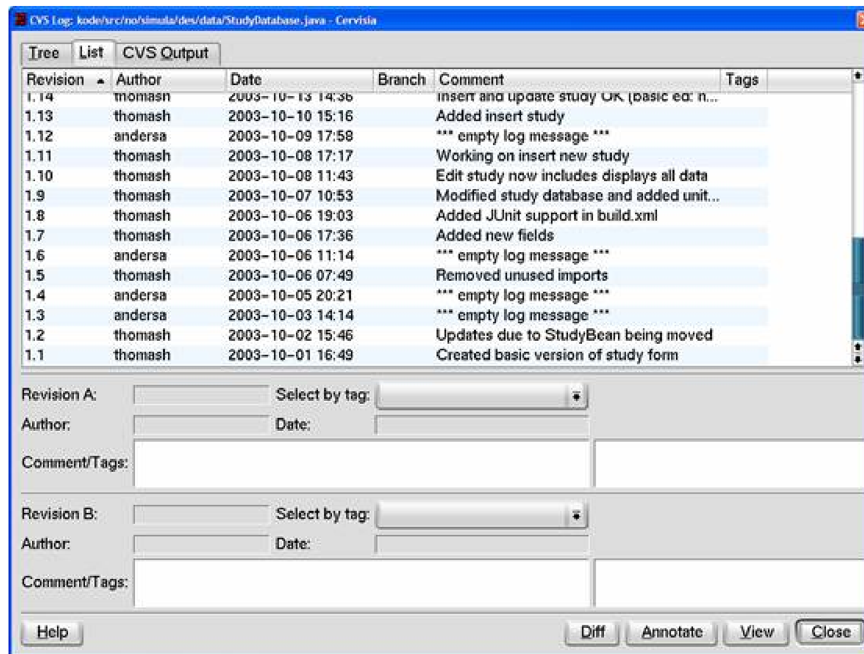
http://subversion.tigris.org/

- Et open-source alternativ til CVS (brukes i prosjektet)
- Noen egenskaper:
  - Most current CVS features. Subversion's interface to a particular feature is similar to CVS's, except where there's a compelling reason to do otherwise.
  - Directories, renames, and file meta-data are versioned. Lack of these features is one of the most common complaints against CVS. Subversion versions not only file contents and file existence, but also directories, copies, and renames.
  - Choice of database or plain-file repository implementations Repositories can be created with either an embedded database back-end (BerkeleyDB) or with normal flat-file back-end, which uses a custom format.
  - Efficient handling of binary files Subversion is equally efficient on binary as on text files, because it uses a binary diffing algorithm to transmit and store successive revisions.
  - All output of the Subversion command-line client is carefully designed to be both human readable and automatically parseable; scriptability is a high priority.

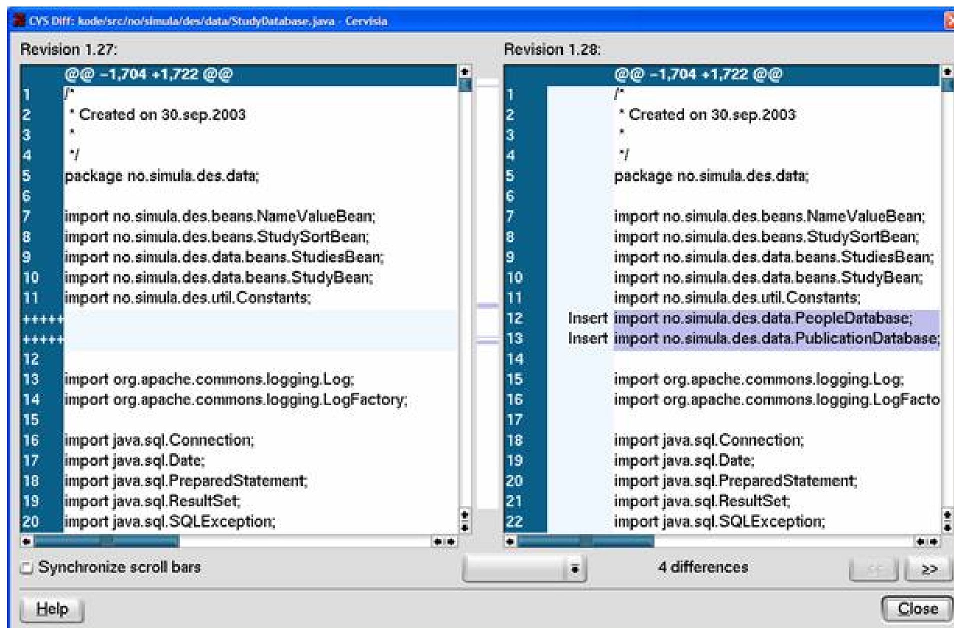
Open-source grafisk grensesnitt mot CVS

- F.eks. ViewCVS: http://viewcvs.sourceforge.net/
  - Support for filesystem-accessible CVS and Subversion repositories.
  - Individually configurable virtual host support.
  - Line-based annotation/blame display (*CVS only*).
  - Revision graph capabilities (via integration with CvsGraph) (*CVS only*).
  - Syntax highlighting support (via integration with GNU enscript).
  - Template-driven output generation.
  - Colorized, side-by-side differences.

Historikken til en fil



Forskjell mellom versjoner



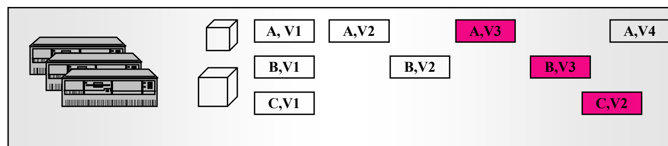
Generasjoner av konfigurasjonsstyringsverktøy

- 1. generasjon: SCCS, RCS, CVS,
- 2. generasjon:
  - Subversion http://subversion.tigris.org/
  - ¨Microsoft® Visual SourceSafe™ msdn.microsoft.com/ssafe/default.asp
  - PVCS www.pvcs.synergex.com
  - Perforce www.perforce.com/
  - Web-baserte overbygninger til CVS

  (Det finnes et utall gratisverktøy og kommersielle verktøy i denne klassen, i tillegg til at en del firmaer 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.

III. Konfigurasjonsdatabasen (repository)



- Konfigurasjons-databasen er sentral og inneholder:
  - Kildekomponenter i ulike versjoner
  - Applikasjonskonfigurasjoner: Sammenstilling av versjoner av kildekomponenter
  - Et delta er en atomisk endring av typisk en enkelt fil med flg. attributter: *filnavn, dato, utvikler, kommentarer*
  - Andre attributter som kan avledes er: *størrelse* (antall linjer lagt til/fjernet), *ledetid* (intervall fra start til fullførelse), *hensikt* (fiksing, nytt)

Konfigurasjonsdatabasen (repository) bør kunne gi informasjon om:

- Hvem arbeider med/er ansvarlig for en bestemt systemversjon eller en bestemt seksjon av systemet?
- 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?

Hva kan man studere basert på konfigurasjonsdatabasen?

Audris Mockus, Avaya Labs Research:

Use project's repositories of change data to model phenomena in software projects

- What makes some changes hard and long?
- What processes/tools work and why?
- Estimation: predict project repair effort from planned new features
- Plan for field problem repair after the release
- Identify release readiness criteria

IEEE Transactions on Software Engineering, 28(7):625–637, July 2002:

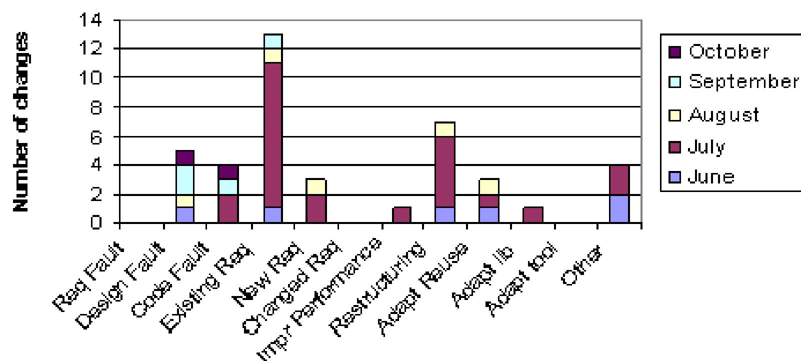
Using Version Control Data to Evaluate the Impact of Software Tools: A Case Study of the Version Editor

David L. Atkins, Thomas Ball, Todd L. Graves and Audris Mockus
Univ. of Oregon, Microsoft Research, Los Alamos National Lab, Avaya Labs Research

ABSTRACT

Software tools can improve the quality and maintainability of software, but are expensive to acquire, deploy and maintain, especially in large organizations. We explore how to quantify the effects of a software tool once it has been deployed in a development environment. We present an effortanalysis method that derives tool usage statistics and developer actions from a project's change history (version control system) and uses a novel effort estimation algorithm to quantify the effort savings attributable to tool usage. We apply this method to assess the impact of a software tool called VE, a version-sensitive editor used in Bell Labs. VE aids software developers in coping with the rampant use of certain preprocessor directives (similar to #if/#endif in C source files). Our analysis found that developers were approximately 40% more productive when using VE than when using standard text editors.

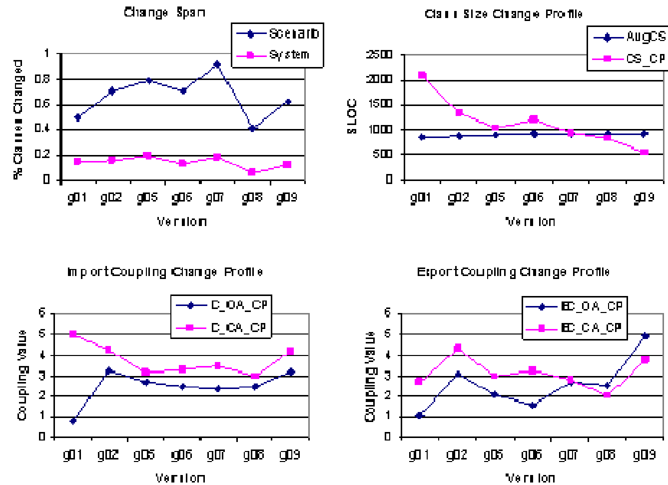
Kategorier av endringer



[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

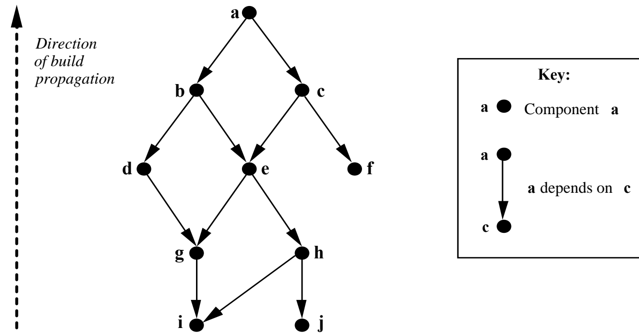


[Erik Arisholm, Empirical Asessement of Changeability in Object-Oriented Software, PhD Thesis, ISSN 1501-7710, No. 143, Unipub forlag, Oslo, February 2001]

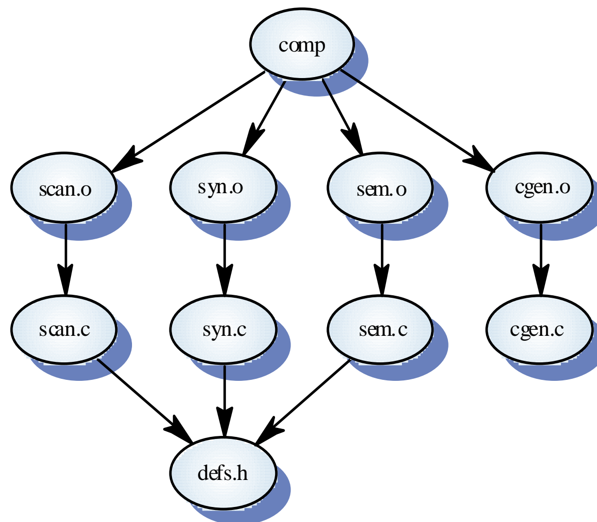
Del IV: Systembygging (build management)

- **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 V: 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++

klasseDefinisjoner.h

```
class Person
{
    void print ();
};

class Konto
{
    void print ();
};
```

printPerson.C

```
#include <klasseDefinisjoner.h>;
void printPerson ()
{
    P = new Person();
    P.print();
}
```

printKonto.C

```
#include <klasseDefinisjoner.h>;
void printKonto ()
{
    K = new Konto();
    K.print();
}
```


Avhengigheter – C++

Person.h

```
class Person
{
    void print ();
};
```

printPerson.C

```
#include <Person.h>;
void printPerson ()
{
    P = new Person();
    P.print();
}
```

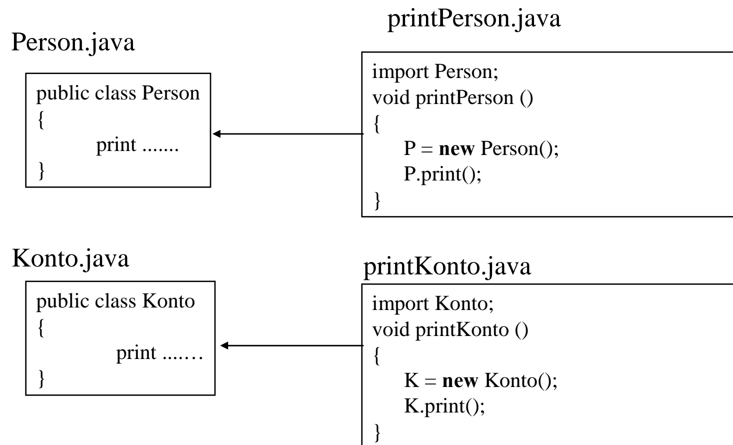
Konto.h

```
class Konto
{
    void print ();
};
```

printKonto.C

```
#include <Konto.h>;
void printKonto ()
{
    K = new Konto();
    K.print();
}
```

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 og inneholdes ofte i SCM verktøy
- 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.

*Se også artikkel I IEEE Software nov/des 2004
** Ant (Another Neat Tool)

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 integrajon? http://c2.com/cgi/wiki?ContinuousIntegration

Del VI: Endringshåndtering generelt

Endringshåndtering omfatter mer enn konfigurasjonsstyring 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)**

[simula . research laboratory]

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 støtter både versjonshåndtering og systembygging**