CASE-verktøy og utviklingsomgivelser

Beskrevet i tilknytning til andre temaer – der verktøy/omgivelser er ment å gi støtte:

Kap. 1.1.9, 4.5, 11.2.1, 13.4, 22.3, 23.4, 29.5

\+ **Programmeringsstandarder (kodekonvensjoner for Java)**

Betydning av verktøy

- **Verktøy har alltid vært viktig for menneskeheten**
- **Verktøy har vært sentralt i informatikken siden 50-tallet**

Mål

- Dere skal ha et bevisst forhold til bruk av verktøy
- God systemutvikling kreves for å lage gode datasystemer og kjennetegnes blant annet gjennom hensiktsmessig bruk av verktøy. Tenk over hvilke verktøy som vil kunne støtte ulike aktiviteter i en konkret systemutviklingskontekst

Tau UML Suite

Vi har valgt et avansert, kommersielt modelleringsverktøy på INF3120:

As of 1 January 2003, Tau was installed at approx. 700 sites worldwide with about 40 000 licenses. The Tau Modeling tools (UML and SDL) had by March 2003 27% of the world market for embedded software modeling tools. Rational Rose was number one with 50%. Telelogic Tau's traditional customer base is in the domain of embedded software.

Hva er ”CASE”?

Computer-aided software engineering:

støtte av dataverktøy til utvikling og vedlikehold av datasystemer

Eks. på eksamensoppgave (H2000)

Eks. på eksamensoppgave (H2000) Oppg. 2D

Angi hvilke systemutviklingsverktøy dere benyttet i prosjektet (oblig). Klassifiser disse verktøyene. Beskriv fordeler og ulemper med bruk av disse verktøyene i forhold til oppgavene som ble løst. (Tips: Angi kriterier du har brukt i evalueringen og hva du sammenligner med når du angir fordeler og ulemper.)

CASE-klassifisering

- **Klassifisering gjør det mulig å vurdere og sammenligne CASE-verktøy**
- **Gjør det enklere å sikre at de ulike utviklingsaktivitetene får nødvendig verktøystøtte**
- **CASE-verktøy kan klassifiseres i henhold til:**
  1. **Funksjonalitet – hvilke funksjoner tilbys**
  2. **Prosess-støtte – hvilke utviklingsfaser støttes**
  3. **Kvaliteten på støtten**
  4. **Kostnader ved innføring og bruk**

1. Klasser av funksjonalitet

Verktøy type	Eksempler	Produkter
Planning tools	PERT tools, estimation tools, spreadsheets, issue-trackers	MS project, Maven wiki
Editing tools	Text editors, diagram editors, word processors	Emacs, MS word, Tau UML, Rose
Change management tools	Requirements traceability tools, change control systems	Bugzilla, Jira
Configuration management tools	Version management systems, system building tools	CVS, SourceSafe, Subversion, Perforce, ClearCase, Make, Tau
Prototyping tools	Very high-level languages, user interface generators	Genova, Visio
Method-support tools	Design editors, data dictionaries, code generators	Tau UML, Rational Rose, ArgoUML, PoseidonUML, MagicDraw, Enterprise Architect, Visio
Integrated Development Environments (IDE)/ Language-processing tools	Compilers, interpreters	Netbeans, Eclipse, Jbuilder, IntelliJ, Forte, Visual Studio, VisualAge, JDeveloper
Program analysis tools	Cross reference generators, static analysers, dynamic analysers	Source Navigator, Purify & Quantify
Testing tools	Test data generators, file comparators	JUnit, Loadrunner, TestDirector, Jira, Unix diff
Debugging tools	Interactive debugging systems	Bugzilla, Jira
Documentation tools	Page layout programs, image editors	
Re-engineering tools	Cross-reference systems, program re-structuring systems	

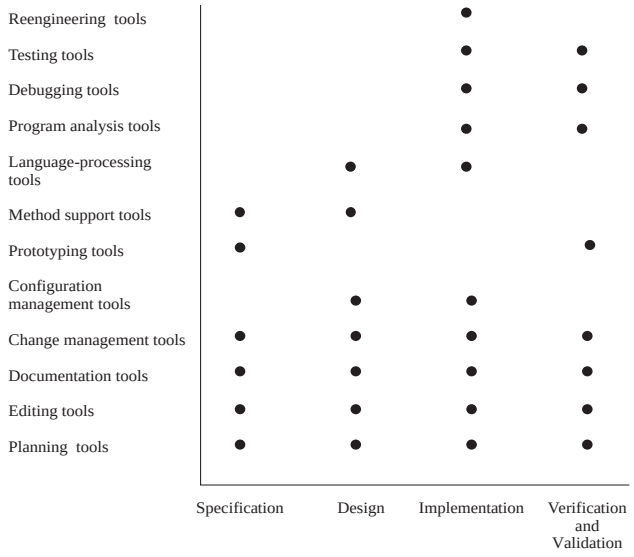
Fire firmaer - samme prosjekt

Kategori	Firma A	Firma B	Firma C	Firma D
Prosjektstyring	MS Project, Word, Excel, Maven	MS Project, Word, Excel		Excel
Designdokumenter	MS Word	MS Word	MS Word	MS Word
Prototyping UI	Visio	n.a.	n.a.	Powerpoint, excel, tavle, digitalkamera
Modeller	Visio	Poseidon for UML	MS Visio/ ArgoUML	Word, Powerpoint, Visio
IDE	IntelliJ/Eclipse	Netbeans	Eclipse	Netbeans/Forte
Build&Deploy	Ant	Ant	Ant	Ant
Versjonskontroll	TortoiseCVS/ CVS	CVS	CVS	CVS
Test/Teststyring	Word	JUnit/cactus	JUnit	Word for test-beskrivelser. Excel for oppfølging av avvik
Annet	Notepad			

Oppgave – andre verktøy? Erfaringer?

Verktøy type	Eksempler	Produkter
Planning tools	PERT tools, estimation tools, spreadsheets, issue-trackers	
Editing tools	Text editors, diagram editors, word processors	
Change management tools	Requirements traceability tools, change control systems	
Configuration management tools	Version management systems, system building tools	
Prototyping tools	Very high-level languages, user interface generators	
Method-support tools	Design editors, data dictionaries, code generators	
Language-processing tools	Compilers, interpreters	
Program analysis tools	Cross reference generators, static analysers, dynamic analysers	
Testing tools	Test data generators, file comparators	
Debugging tools	Interactive debugging systems	
Documentation tools	Page layout programs, image editors	
Re-engineering tools	Cross-reference systems, program re-structuring systems	

2. Verktøy for ulike utviklingsfaser



Sommerville figur 4.15

Valg av verktøy

- **Hvilke verktøy (og metoder og teknikker) skal vi anvende i vår bedrift?**
- **Valget vil avhenge av:**
  - **type oppgaver på hvilken type applikasjon som skal utføres**
  - **type utviklere (kompetanse, erfaring, utdanning)**
  - **type omgivelser (organisasjon, teknologisk miljø)**
  - **kostnader**

3. Kvalitet på verktøy

- Graden av støtte for de ulike funksjonene/aktivitetene
- Hvor enkelt å lære og bruke når det først er lært
- Brukergrensesnitt
- Stabilitet, feil
- Dokumentasjon/"hjelp-funksjoner"/brukermanualer/ leverandørstøtte
- Leverandørstabilitet/oppdateringer/nye versjoner
- Det viktigste: hvor godt blir sluttproduktet og hvor lang tid tar det å utvikle det?

Merk: Et verktøy er at datasystem og vil være gjenstand for de samme kvalitetskriterier som andre datasystemer, jfr forelesningen 29.9.05

Metoder for evaluering av kvalitet

Generelt ikke trivielt å angi kvalitet, krever empiri/ systematisk erfaring fra bruk og vil være relativ til:

- type oppgaver på hvilken type applikasjon som skal løses
- type utviklere (kompetanse, erfaring, utdanning)
- type omgivelser (organisasjon, teknologisk miljø) .

Behov for å gjennomføre ulike typer studier

- Case-studier
- Kontrollerte eksperimenter
- Spørreundersøkelser (surveys)

Case-studie i ABB: Erfaringer med Rational Rose

"Rational Rose was considered an adequate tool, but the interviewees found that it lacked stability, that it was particularly difficult to edit sequence diagrams and that they did not manage to specify interfaces or check consistency using the tool. Some felt that Rational Rose was basically used as a drawing tool, and that all possibilities in the tool were not exploited. Others felt that they had managed to make diagrams in Rational Rose, but that the diagrams were difficult to read."

[Bente Anda, Kai Hansen, Ingolf Gullesen and Hanne Kristin Thorsen, Experiences from Using a UML-based Development Method in a Large Organisation, To appear *Journal of Empirical Software Engineering*, 2005]

Kontrollert eksperiment A

- 130 innleide konsulenter fra 8 firmaer: " … each subject used a Java development tool of their own choice, e.g., JBuilder, Forte, Visual Age, Visual J++, and Visual Café."

→ Ingen signfikant forskjell i utviklingstid eller kvalitet mht verktøy.

[Erik Arisholm and Dag Sjøberg, Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software, *IEEE Transactions on Software Engineering*, 30(8), pp. 521-534, August 2004]

Kontrollert eksperiment B

- Fire CASE tools (Systemator, PowerBuilder, AdsOnline, Cobol) evaluert på Telenor:

→ Utvikling: PB 1.0 FP pr time, Systemator 0.7, AdsOonline 0.4, Cobol 0.25

→ LOC pr time omtrent det samme, men dårlig mål siden ulik funksjonalitet pr LOC

→ Merk at må måle effekt både for nyutvikling og for vedlikehold

[Magne Jørgensen and Sigrid Steinholt Bygdås, An empirical study of the correlation between development efficiency and software development tools, *Telektronikk*, vol. 95, no. 1, 1999, pp. 54-62]

Kontrollert eksperiment C

[Bente Anda and Dag Sjøberg, Applying Use Cases to Derive versus Validate Class Diagrams, *To appear in Empirical Software Engineering, 2005*]

Merk: liten oppgave, få utviklere i noen av gruppene, ulike metoder og studenter for pen&papir

The SAS System 11:56 Sunday, October 3, 2004

Tool	Obs	Variable	Minimum	Median	Maximum
Pen&papir	21	Completeness	1.0	3.0	5.0
		Structure	1.0	3.0	5.0
Tau	26	Completeness	0.0	3.0	5.0
		Structure	0.0	3.0	5.0
MagicDraw	3	Completeness	3.0	5.0	5.0
		Structure	1.0	2.0	5.0
Visio	11	Completeness	0.0	4.0	5.0
		Structure	2.0	3.0	5.0
RationalRose	6	Completeness	5.0	5.0	5.0
		Structure	1.0	2.5	4.0

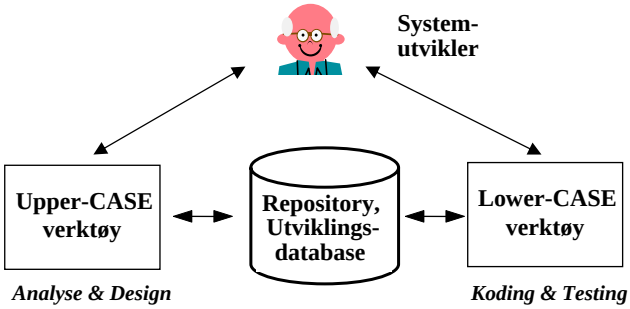
Analysis Variable : Time

Tool	Obs	Minimum	Mean	Maximum
Pen&papir	21	124	176	264
Tau	26	136	214	264
MagicDraw	3	230	250	271
Visio	11	126	194	262
RationalRose	6	229	287	352

4. Verktøykostnader

- Innkjøp/lisenser
- Nye versjoner/oppgraderinger
- Opplæringskostnader

Generell arkitektur for CASE-verktøy



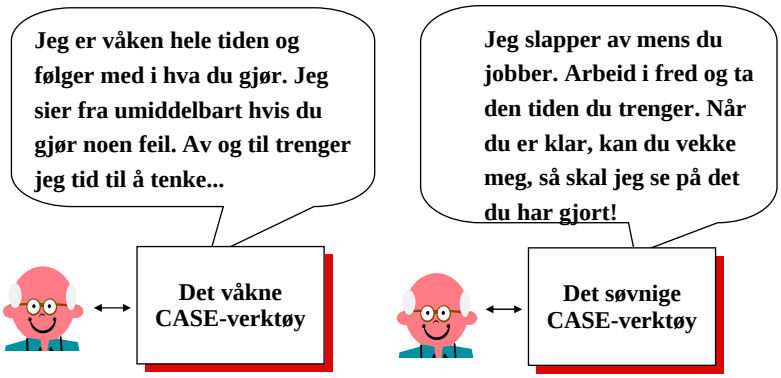
Modelleringsverktøy

Integrerte utviklingsomgivelser

Hva kan ligge i utviklingsdatabasen?

- Spesifikasjoner for systemet
  (Use Case-diagrammer, Klassediagrammer, Sekvensdiagrammer, Dataflytdiagrammer? Strukturdiagrammer? Programflytdiagrammer? …)
- Argumentasjon for spesifikasjonene
- Spesifikasjoner i flere versjoner (konfigurasjonsstyring?)
- Dokumentasjon utover spesifikasjonene
- Overordnede brukerkrav
- Tester
- Prosjektplaner, milepælsdefinisjoner
- Etc.

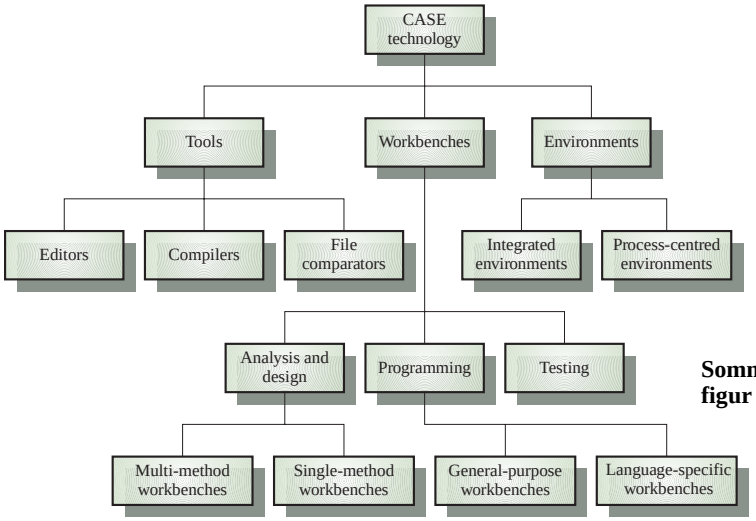
Våkne og søvnige verktøy



Eksempel i Tau UML

- **"Våken" syntakssjekking**
  - **En kontinuerlig sjekking som gjør at man overholder reglene for korrekte diagrammer**
- **En egen sjekk-funksjon som gjelder modellen/pakken/ systemet. Man kan da velge ulike nivåer av sjekking:**
  - **Bare ett diagram**
  - **hele modellen**
  - **sekvensdiagrammer mot klassediagrammer**

Verktøy, arbeidsbenker, utviklingsomgivelser



Sommerville figur 4.16

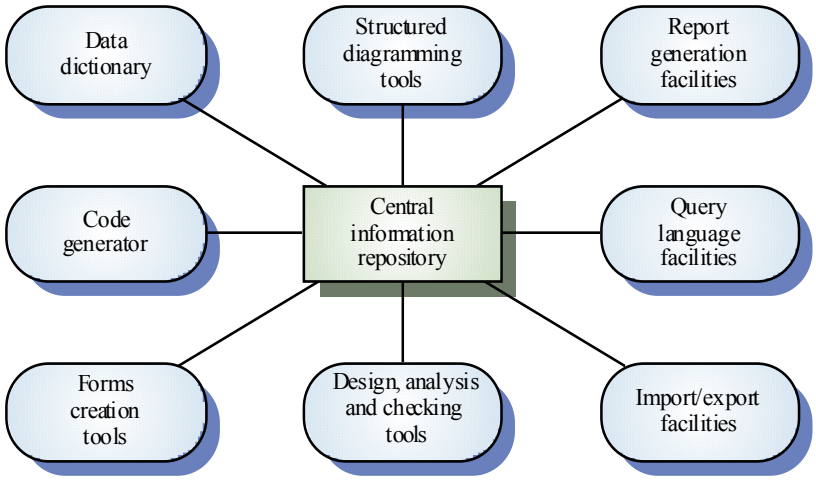
Merk: mye ulike språkbruk her

CASE-arbeidsbenker/omgivelser

Verktøysamling som gir omfattende støtte i bestemte faser av systemutviklingen. En kjerne tilbyr felles tjenester til alle verktøy og en viss grad av dataintegrasjon

- **Arbeidsbenker for analyse og design**
- **Arbeidsbenker for programmering, dvs. programmeringsomgivelser (Unix, språkorienterte, 4GL, …)**
- **Arbeidsbenker for testing**
- **Meta-CASE-arbeidsbenker**

Analyse og design-arbeidsbenk/modelleringsverktøy

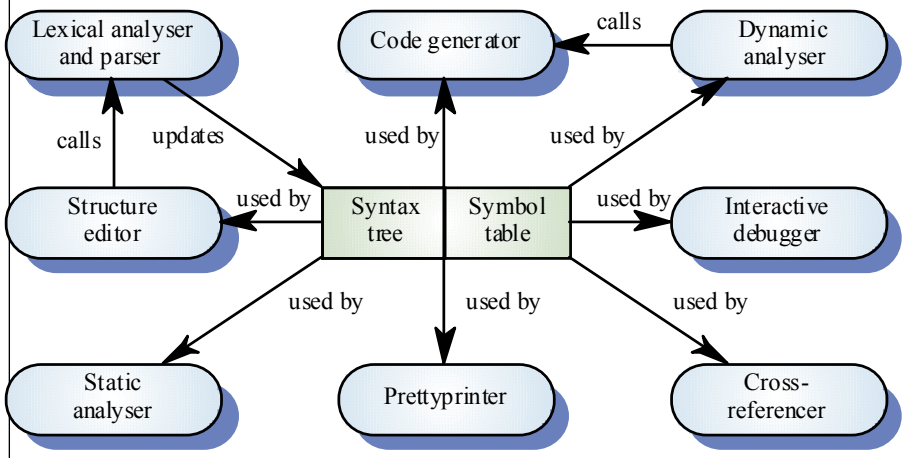


Modelleringsverktøy

- ArgoUML - http://argouml.tigris.org/ opensource
- PoseidonUML - http://www.gentleware.com/
- MagicDraw - http://www.magicdraw.com
- Enterprise Architect - http://www.sparxsystems.com.au/
- TAU UML
- Rational Rose XDE
- Jbuilder her også, siden de faktisk har UML-modellering.
- Genova
- OptimalJ, prøver å implementere OMG's MDA-standard

Skillet mellom programmeringsverktøy (IDE'er) og modelleringsverktøy er i ferd med å viskes ut, for eksempel Rational XDE er plug-in til bl.a. Eclipse

Programmeringsomgivelse (IDE)



Programmeringsomgivelser (IDE)

- Eclipse og Netbeans er to gratis, open source ide'er. IBM supporterer Eclipse, Sun supporterer Netbeans. Begge har kommersielle versjoner av verktøyene
  - Netbeans http://www.netbeans.org/products/ide/
  - Eclipse www.eclipse.org, http://www-106.ibm.com/developerworks/library/os-ecjbuild/?ca=drs-tp3704
- Av kommersielle verktøy er de ledende:
  - JBuilder (Borland) - http://www.borland.com/jbuilder/pdf/jb2005_feature_matrix.pdf |
  - IntelliJ (jetbrains) - http://www.jetbrains.com/idea/features/
  - JDeveloper (Oracle) – http://www.oracle.com/technology/products/jdev/htdocs/jdev9irc_fo.html1|3w1

Statisk analyseverktøy

Statisk analyse: analyse av kildekode i henhold til regler

- Syntaksanalysatorer/kompilatorer
- Programvarestandarder
- Etc.

Dynamisk analyse: analyse av programutførelser

- Minne
- Ytelse
- Hvilke deler av et program som faktisk brukes, og hvor ofte
- Etc.

Eksempel på statisk analyse:
Sjekk av programvare-standarder (kap. 24.1)

- **Bidrar til kvalitetssikring**
- **Kan være internasjonale, nasjonale eller spesifikke til organisasjon eller prosjekt**
- **Produkt-standarder definerer egenskaper ved alle komponenter, *f.eks.* *kodestandarder*. Sjekking av slike er eksempel på *programanalyse***
- **Prosess-standarder definerer hvordan programvareprosessene skal gjennomføres**

Kodekonvensjoner/standarder

- **Navngivning**
- **Deklarasjoner**
- **Statements**
- **Innrykk & blanke**
- **Filorganisering**
- **Dokumentasjon**
- **etc.**

For eksempel Java kodekonvensjoner: http://java.sun.com/docs/codeconv/

→ *ta en titt!*

Verktøystøtte for Java kodekonvensjoner

- **Telelogic LogiScope (http://www.telelogic.com/products/Logiscope/)**
- **QJ-Pro (http://qjpro.sourceforge.net/)**
  - **Inneholder en rekke sjekker som kan slås av og på, f. eks.**
    - bad programming practices,
    - conformance to coding standards,
    - misuse of the Java language,
    - code structure and
    - potential bugs at the earliest stages of development.
  - **Men mangler muligheten for å definere egne regler**

[simula . research laboratory]

Inspeksjon: Automatisert statisk kodeanalyse

- Programmer som analyserer kildekode og avdekker avvik fra spesifiserte krav
  - Sjekker at alle klassenavn begynner med store bokstaver
  - Sjekker at metoder ikke er for lange
  - Sjekker at input-paramtere valideres
  - Sjekke at alle metoder er dokumentert
  - Ikke-eksekverbar kode
  - Variabler som aldri brukes
  - Minnehåndtering
- Kan integreres i utviklingsmiljøet, versjonshåndteringssystemet eller byggesystemet
  - For eksempel kan kode hvor verktøyene rapporterer feil i nektes innsjekking
- Raskt å kjøre (billig) i motsetning til inspeksjon
  - Men mange ting fanges ikke opp her....
- Eksempel på verktøy:
  - Kompilatorer, IntelliJ, jDepend, jMetric

Hvorfor?

- 80% of the lifetime cost of a piece of software goes to maintenance. Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

Copyright 1995-1999 Sun Microsystems

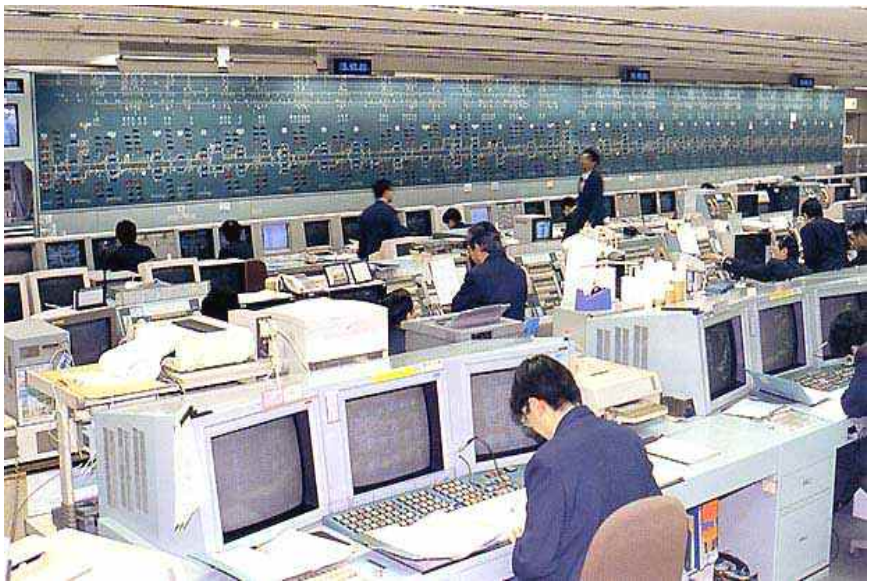
- Faster development since the original programmer is the first to benefit from the improved readability.
- Better cooperation within the development team since everybody can read and understand everybody else's code.
- Fewer bugs since the code becomes easier to understand. It is also more likely that the code will be read and understood by more people if it is easy to do so. More eyes on the code means more bugs discovered.
- The produced software will be easier to maintain since it isn't dependent on the original developer.

Jens Gustavsson

Behovet for regler og konvensjoner

Shinkansen

Kontrollsenteret



Kontroll og disiplin

- Every Bullet Train in Japan is controlled by computers. The computers are kept in a building called the CTC (Central Traffic Control) in Tokyo. The cab of a Shinkansen is connected to computers that operate the train. The automatic train control (ATC) system enhances safety by preventing trains from colliding with each other. If a train is going too fast, the brakes operate automatically.
- Tokyo-Osaka (500 km): 285 trains a day carrying 360,000 passengers. In 36 years, there has not been a single death due to a train accident. As for reliability, the average delay is 0.4 minutes (average delay on arrival for all trains). Most of this delay is due to natural phenomena such as heavy rain, typhoons and snowfall.
- Shinkansen maintenance system
  At night after train operation has finished, fine adjustments are made to rail positions. Also, rails and overhead catenary showing signs of wear and deteriorated ballast are replaced. During operation, a multiple inspection train (called Doctor Yellow) is run once every 10 days to make a precision check of the condition of the tracks, overhead power catenary, signals and communications. The maintenance of cars consists of daily inspections focusing mainly on the replacement of worn parts, regular inspections which mainly check functions, dismantling inspections for bogies and a general overhaul which includes the inspection of the car body. These inspections are conducted on the basis of kilometers traveled or at set time intervals.

Eks.: Japansk lokfører sovnet i 270

- Mens det japanske toget rullet jevnt og rolig videre, sovnet lokføreren stille og fredelig bak spakene. Hastighetsmåleren viste 270 kilometer i timen, opplyste polititalsmenn torsdag.
- Mannen våknet etter åtte minutter da systemet for automatisk togstopp stanset høyhastighetssettet av typen Shinkansen på en stasjon i provinsen Okayama sør i Japan. De rundt 800 passasjerene om bord i toget merket ingen ting til episoden.
- (NTB 27.02.03 kl. 07:31)

Kryssreferanse-verktøy

- Enkle kryssreferanseverktøy lager lister med navn og type på *identifikatorer* som er brukt i et program. Hver linje som deklarerer et navn, angir hvor en variabel etc. er brukt, og hver linje hvor et navn er brukt, inneholder en referanse til hvor det er deklarert.
- Avanserte kryssreferanseverktøy for store datasystemer har en egen database som inneholder kildekode-informasjon og har et tilhørende kommando eller spørrespråk man kan bruke til å hente fram og analysere den statiske informasjonen. Databasen blir gjerne oppdatert når man kompilerer (hvis parameter satt).
- Noen verktøy har avanserte brukergrensesnitt med fargede grafer (feil angitt i rødt, ulike typer bokser for ulike typer objekter etc.).
- Stor nytte for store systemer!
- I dag ofte integrert med syntaksbaserte editorer

Source Navigator

http://sources.redhat.com/sourcenav/

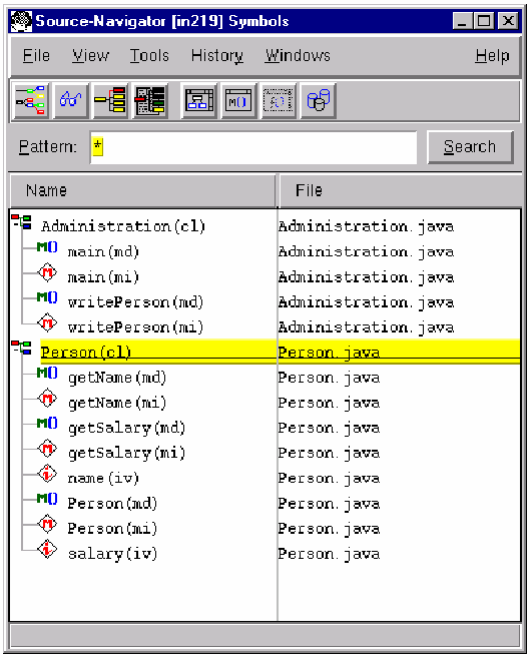
To klasser i Java

`Person.java:`

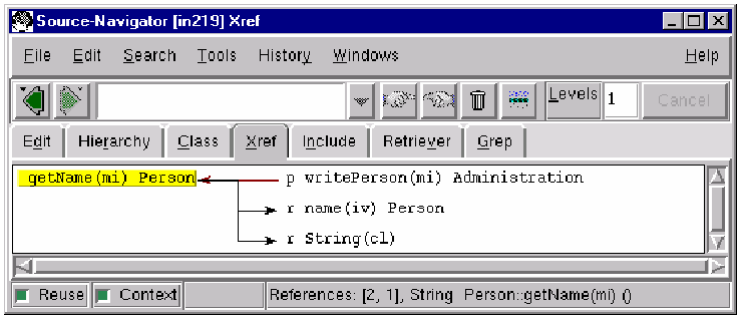
```
class Person {
  public String name;
  public int salary;
  public Person (String n, int s) {
     name = n;
     salary = s;
   }
  public String getName() { return name; }
  public int getSalary() { return salary; }
}
```

`Administration.java:`

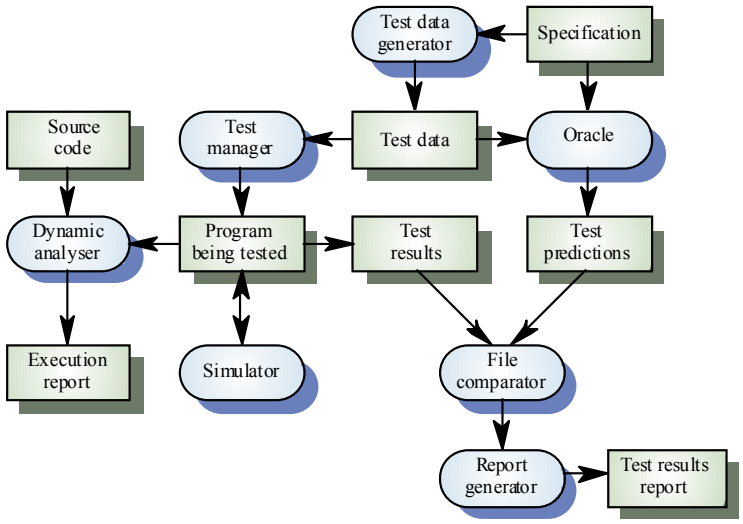
```
class Administration {
  public static void main (String args[]) {
    Person p = new Person("Ola Olsen",30000);
    writePerson(p);
  }
    static void writePerson (Person p) {
    System.out.println("Name :"+p.getName());
    System.out.println("Salary :"+p.getSalary());
    }
}
```

Cross-reference for selected item (getName)



Testverktøy (kap. 23.4)



Sommerville figur 23.17

Verktøy for teststyring og automatisert testing

- JUnit har slått an for lavnivå java enhetstesting
- Loadrunner for stresstesting
  http://www.mercury.com/us/products/performance-center/loadrunner/
- Avanserte teststyringsverktøy:
  - TestDirector http://www.mercury.com/us/products/quality-center/testdirector/
  - jira (www.atlassian.com)
- Men i mange prosjekter holder det å bruke bugtracking-verktøy ala bugzilla
- For verktøy for prosjektstyring, kravhåndtering, bugtracking og teststyring, samt eventuelt support-system under forvalting kommer man langt med verktøy som jira.
- Egne verktøy for hvert av disse områdene er sjelden vellykket på grunn av tette sammenhenger mellom disse disiplinene.

Meta-CASE

- Analyse og design-arbeidsbenker fra ulike leverandører er begrepsmessig nokså like. Ofte ligger forskjellen bare i diagram-typene og (reglene og retningslinjene i) metoden som støttes
- Meta-CASE-arbeidsbenker er verktøy som gir assistanse i å generere CASE-arbeidsbenker
- Prinsippet bak Meta-CASE er å parameterisere det spesifikke
  - Programmeringsarbeidsbenker er integrert rundt en syntaks-representasjon som kan defineres separat
  - A & D-arbeidsbenker kan lages ved å bruke et metode-definerings-språk for å definere regler og retningslinjer for en gitt utvklingsmetode

Merk: Lite relevant i dag pga. for komplekst for brukere og dyrt produkt

Verktøy-integrasjon

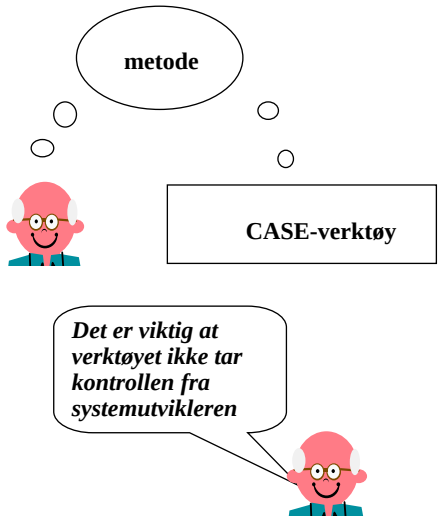
- **Uhensiktsmessig å ha ett verktøy som gjør alt (for stort, for tungt, lite fleksibelt) Dessuten finnes ikke noe slikt på markedet...**
- **Bedre å kunne integrere mindre, spesialiserte verktøy (systemutvikleren kan gripe til det verktøyet som egner seg best - på samme måte som en god håndverker). F.eks.**
  - **Integrering av design-verktøy med dokumentasjonsverktøy**
  - **Integrering av verktøy for analyse, design og programmering med konfigurasjonsstyringsverktøy (jfr. Tau UML)**
- **Men hvordan integrere?**

Dataintegrasjon

- **Felles filer**
  - **Verktøy kommuniserer gjennom et felles filformat. Enkel og den vanligste måten å utveksle informasjon på (f.eks. "piping" i Unix)**
  - **Krever at verktøyene kjenner beskrivelsen av et felles filformat eller at det eksisterer oversettere fra ett filformat til et annet**
  - **Standarder: CDIF, XMI**
- **Felles datastrukturer**
  - **Verktøy kommuniserer gjennom en intern representasjon av noen felles begreper (lukket)**
- **Felles repository**
  - **Verktøy er integrert rundt et objekt-håndteringssystem (OMS) som inkluderer et åpent tilgjengelig skjema med felles begreper**

Prosessintegrasjon (utviklings/vedlikeholdsprosess)

- En eksplisitt modell av utviklingsprosessen i et firma må defineres. (Prosessen bør ledes istedenfor detaljstyres av denne modellen.)
- CASE-systemet har iboende kunnskap om prosessaktiviteter i systemutviklingen – deres faser, regler og verktøyene som brukes. Implisitt finnes en innebygget en modell for systemutviklingen
- CASE-systemet må støtte bedriftens prosessmodell



Åpne CASE-verktøy

- Dataintegrasjonsprotokoller er allment tilgjengelige. Brukerne kan derfor inkludere nye verktøy
- Fordeler
  - Verktøysamlingen kan skreddersys til spesielle organisasjonelle behov
  - Filer som produseres kan håndteres av et konfigurasjonsstyringssystem
  - Inkrementell (skrittvis) innføring og evolusjon (endringer og utvidelser) er mulig
  - Organisasjonen kan ha ulike verktøy fra flere leverandører
- Rational Add-on

Lukkede CASE-verktøy

- Mange kommersielle og eksperimentelle CASE-verktøy er lukkede systemer, f.eks. Tau UML. Kontroll- og dataintegrasjon baseres på interne (skjulte) mekanismer.
- Lukkede arbeidsbenker er vanligere enn åpne
- De tilbyr oftest tett integrasjon, inkludert et felles ”look and feel”
- Det er imidlertid umulig å inkludere tredjeparts verktøy, og brukeren er dermed bundet til en bestemt leverandør

Software Engineering Environments

- En SE-omgivelse (SEE) er en mengde maskinvare- og programvare-verktøy som støtter hele utviklingsprosessen fra første spesifikasjon til levering av endelig system
- Fasilitetene til omgivelsene er integrerte. Omgivelsene bør tilby integrasjon av plattform, data, presentasjon, kontroll og prosess
- Omgivelsene er designet for å støtte en rekke programutviklingsaktiviteter. Dette inkluderer team-baserte aktiviteter hvor f.eks. konfigurasjonsstyring er sentralt|w

Merk: Lite relevant i dag pga. enorme kostnader i utvikling og liten suksess i praksis

CASE-verktøy - oppsummering

- **Ultimate mål: Høy kvalitet på det ferdige datasystemet krever hensiktsmessig bruk av verktøy**
- **Enklere oppsetting, integrasjon, vedlikehold, gjenbruk og kontroll av systemspesifikasjoner og kode**
- **Automatisk kodegenerering**
- **Innføring og sjekking av standarder og konvensjoner**
- **Bedre prosjektstyring og kostnadskontroll**
- **Metodesterke verktøy kan være til god hjelp, men også medføre «metodeimperalisme» - frysing av metoder og teknikker**
- **Leverandøravhengighet**
- **Verktøy krever kompetanse**
- **Verktøy er ikke gratis**
- **Som ansvarlig utvikler/prosjekt-leder/leder bør dere kunne vurdere hvilke verktøy som er hensiktsmessige for et bestemt prosjekt/bedrift**