Lecture 4: Software design

- Review: Software development method and process
- Modelling techniques
- Design/ programming rules
- Model driven architecture

Review: language theory

Explain (each with one sentence):
alphabet, syntax, grammar, semantic.

Draw a short graphic which shows the sets of Chomsky Hierarchy and give as well the type and name of language types.

Give five programming paradigms and explain them?

Software methodology: quality aims

And the quality aims are:

- Reliability/Robustness
- Portability/Scalability
- Usability/Functionality/Correctness
- Maintainability
- Compactness
- Re-usability/Modularity
- Comprehensibility/Understandability
- Schedulability/Efficiency/Flexibility
- Testability
- Security …

Software methodology

There are a lot of techniques which can help to improve the developement of software

Software methodology :=

software development process

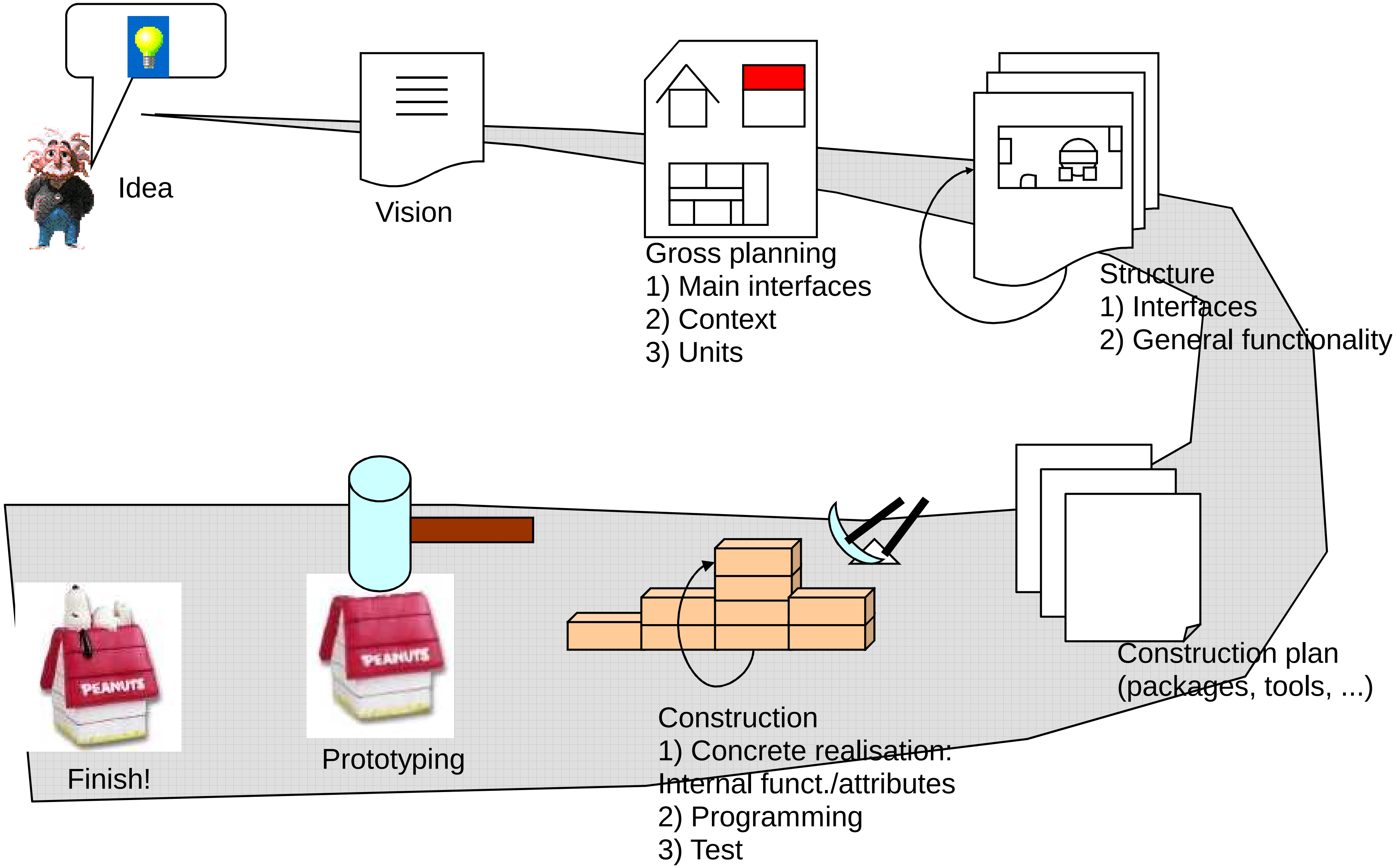
+

process attending modelling techniques

+

(process attending design/ programming rules)

Software development process



Idea

Vision

Gross planning
1) Main interfaces
2) Context
3) Units

Structure
1) Interfaces
2) General functionality

Construction plan
(packages, tools, ...)

Construction
1) Concrete realisation:
Internal funct./attributes
2) Programming
3) Test

Prototyping

Finish!

Lecture 4: Software design

✓- Review: Software development method and process

- Modelling techniques
- Design/ programming rules
- Model driven architecture

Modelling techniques: UML

Unified Modelling Language (UML) =

Notation: informal, graphical representation of a design

+

Metamodel: mathematical, formal correctness of the description

- UML is a standardized language for visualisation, specification, construction and documentation of complex software systems in the field of object oriented solutions
- UML is technology and process independent and usable during the whole development process
- UML is a combination of several technique concepts

Aim: rigorous specification and design

See: *Fowler, Martin; Scott, Kendall: UML konzentriert. Addison-Wesley 2000 (Orig.: UML Distilled. Second edition*
See: http://casablanca.informatik.hu-berlin.de/database/MDA-UML/VL/V2-MODSOFT-I.2.UML-Ueberblick.pdf, Download 07.11.2006

Modelling techniques: UML

Decomposition of a system:

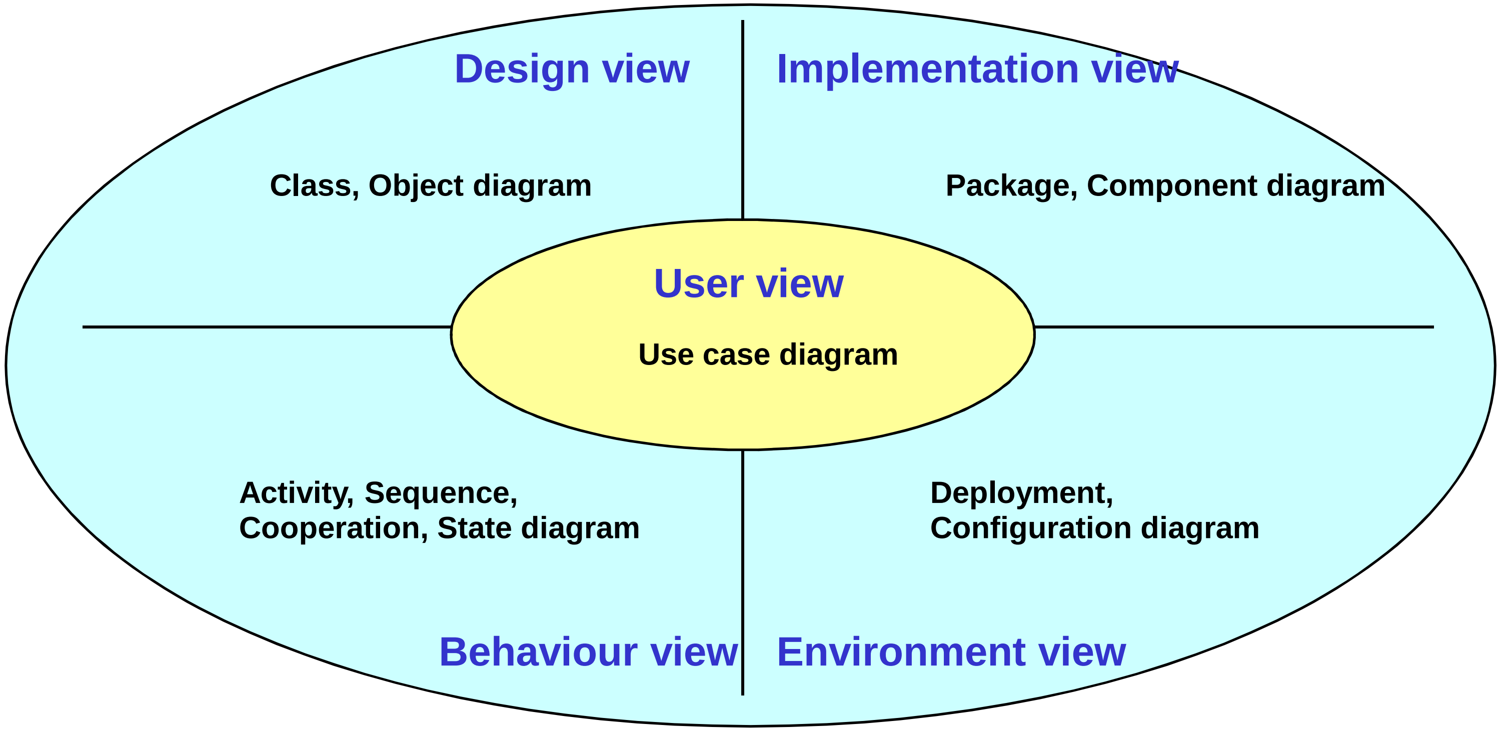
ASSEMBLY INSTRUCTIONS
TICKER
STICK-ON STRIPES
DO-IT-YOURSELF CAT KIT
LEG-REAR OFFSIDE
PURRER
MIAUWER

Grady Booch

(Reference: *http://casablanca.informatik.hu-berlin.de/database/MDA-UML/VL/V2-MODSOFT-I.2.UML-Ueberblick.pdf*, Download 07.11.2006)

Modelling techniques: UML

The basics:



Design view

Class, Object diagram

Implementation view

Package, Component diagram

User view

Use case diagram

Activity, Sequence,
Cooperation, State diagram

Deployment,
Configuration diagram

Behaviour view

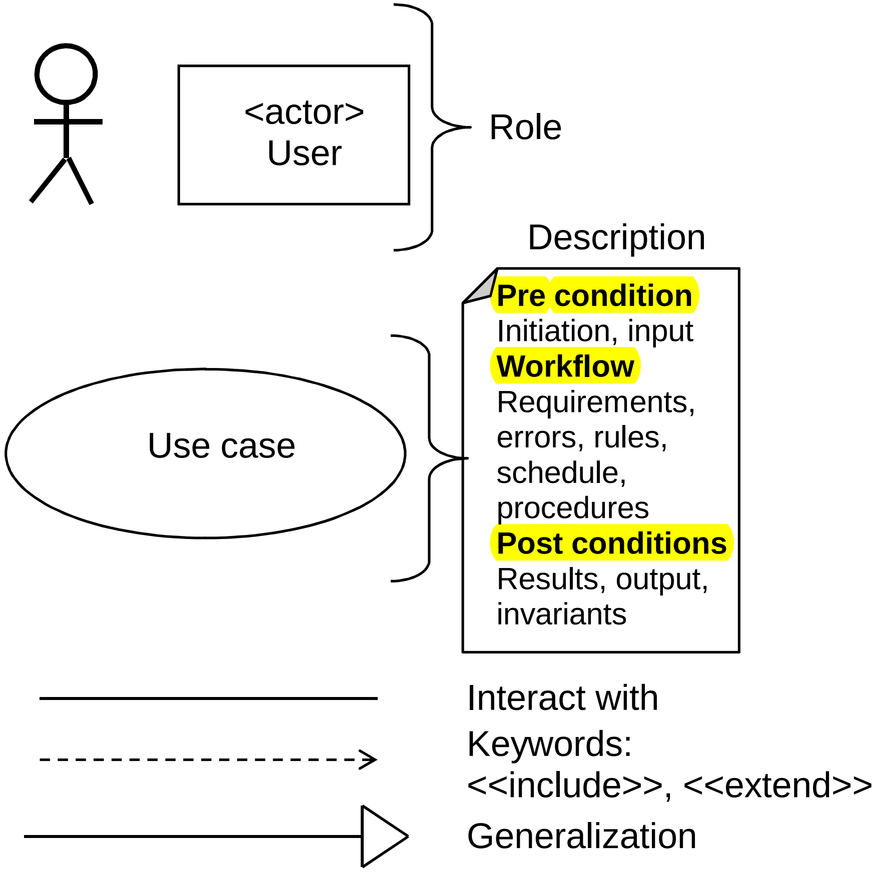
Environment view

See: http://casablanca.informatik.hu-berlin.de/database/MDA-UML/VL/V2-MODSOFT-I.2.UML-Ueberblick.pdf, Download 07.11.2006

Modelling techniques: UML

e.g.: Use case diagram:

Description of the external view of a system (interaction with the environment)



<actor>
User

Role

Description

Pre condition
Initiation, input
Workflow
Requirements, errors, rules, schedule, procedures
Post conditions
Results, output, invariants

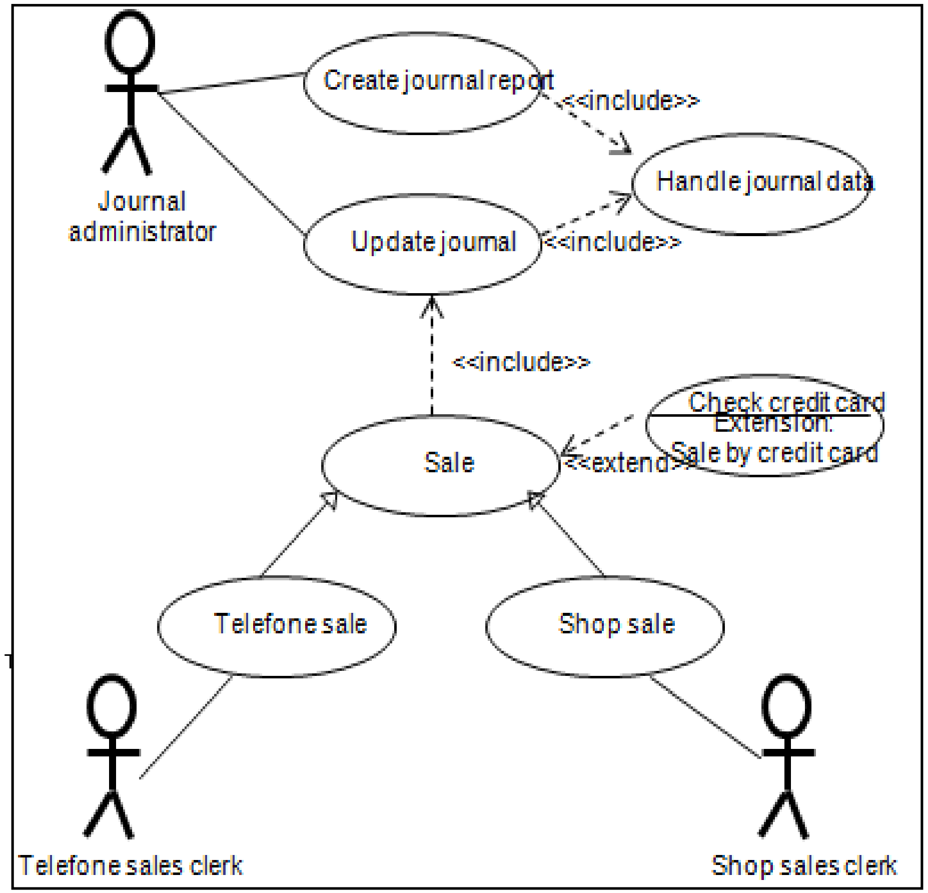
Use case

Interact with

Keywords:
<<include>>, <<extend>>

Generalization

Example:

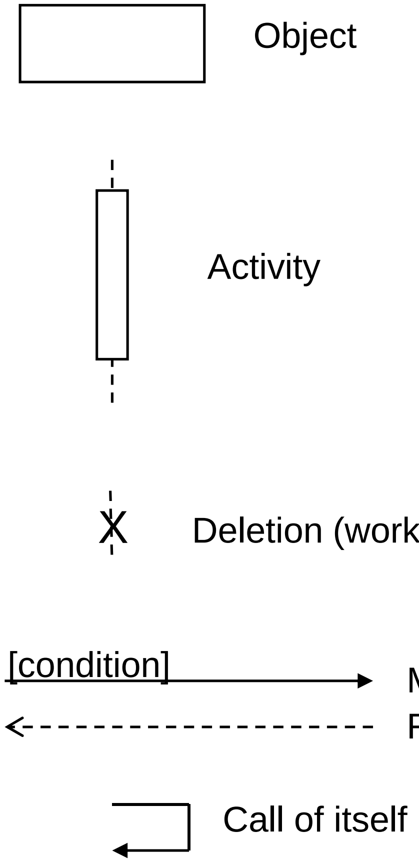


See: *Fowler, Martin; Scott, Kendall: UML konzentriert. Addison-Wesley 2000 (Orig.: UML Distilled. Second edition)*
See: *Oesterreich, Bernd: Analyse und Design mit UML 2. Oldenbourg Wissenschaftsverlag GmbH 2005*
See: *Roff, Jason T.: UML: A Beginner's Guide, McGraw-Hill Companies 2003*

Modelling techniques: UML

e.g.: Sequence diagram:

Description of a dynamic behaviour and interactions for a specific schedule



Object

Activity

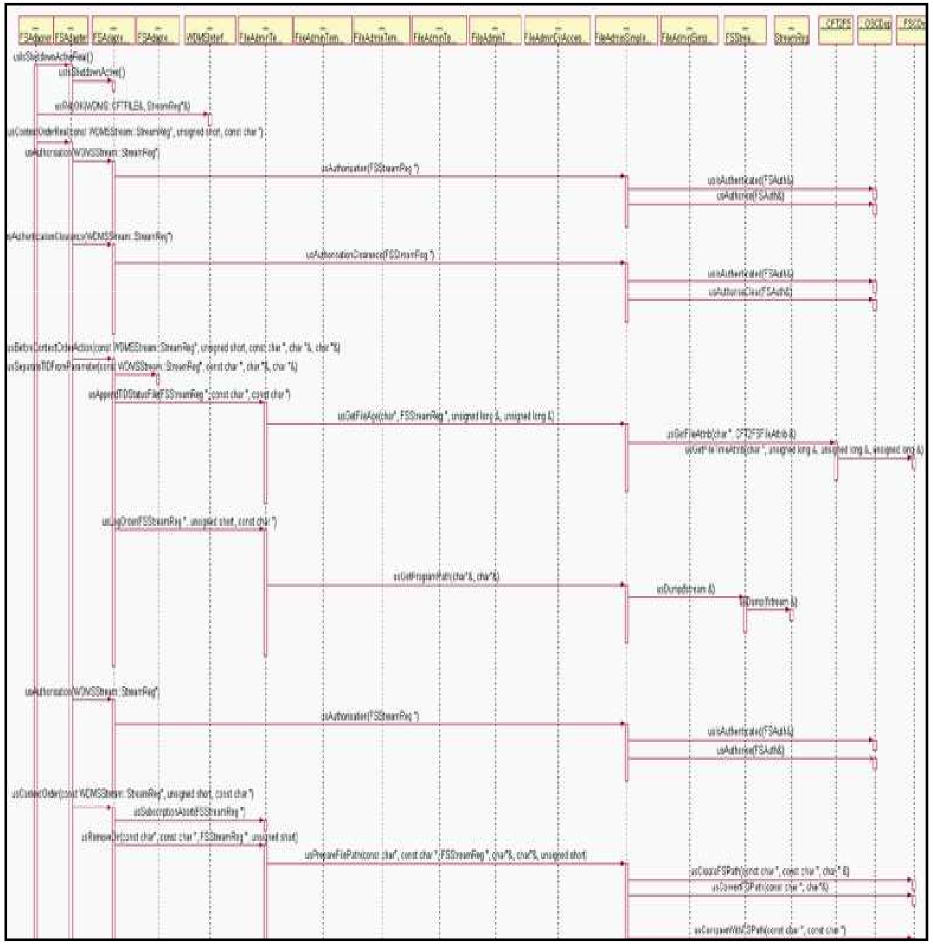
Deletion (workflow stopped)

[condition] Message

Return

Call of itself

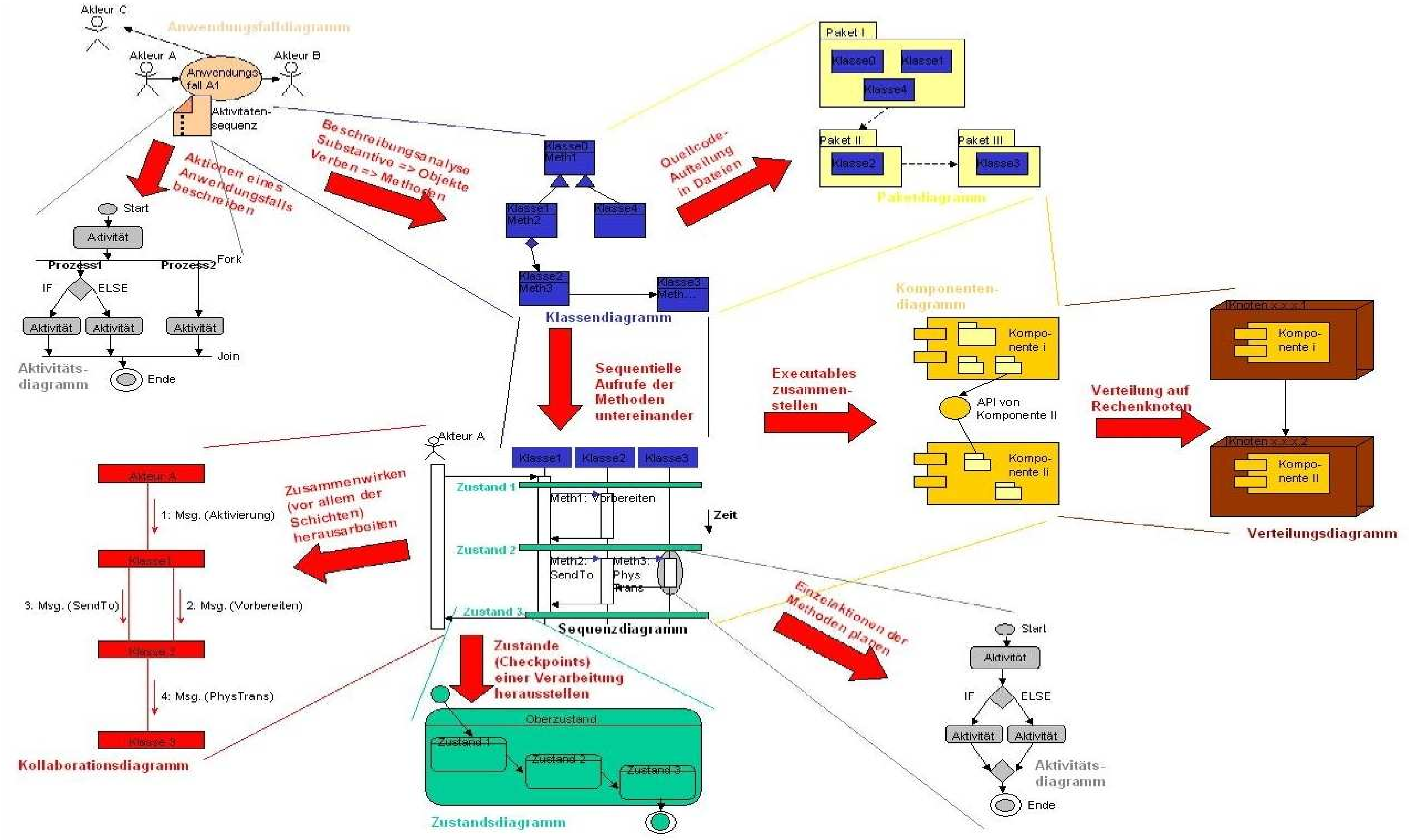
Example:


FSAdapter
WDMSInterf
FileAdminTe
FileAdmin
FSStrea
StreamReq
CFTDFS
OSCDsg
utsAuthorisation(FSStreamReq *)
utsIsAuthenticated(FSAuth&)
utsAuthorize(FSAuth&)
utsAuthorisationClearance(FSStreamReq *)
utsAppendTDStatusFile(FSStreamReq *, const char *, const char *)
utsGetFileAge(char *, FSStreamReq *, unsigned long &, unsigned long &)
utsGetProgramPath(char *&, char *&)
utsDumpStream(&)
utsPrepareFilePath(const char *, const char *, FSStreamReq *, char *&, char *&, unsigned short)

See: *Fowler, Martin; Scott, Kendall: UML konzentriert. Addison-Wesley 2000 (Orig.: UML Distilled. Second edition)*

See: Neidhardt, Alexander: *http://mediatum.ub.tum.de/mediatum/servlets/TUMDistributionServlet?id=mediaTUM_derivate_000000000002301*, Download 12.12.2006

Modelling techniques: UML



See: Neidhardt, Alexander: *http://mediatum.ub.tum.de/mediatum/servlets/TUMDistributionServlet?id=mediaTUM_derivate_000000000002301*, Download 12.12.2006

Design

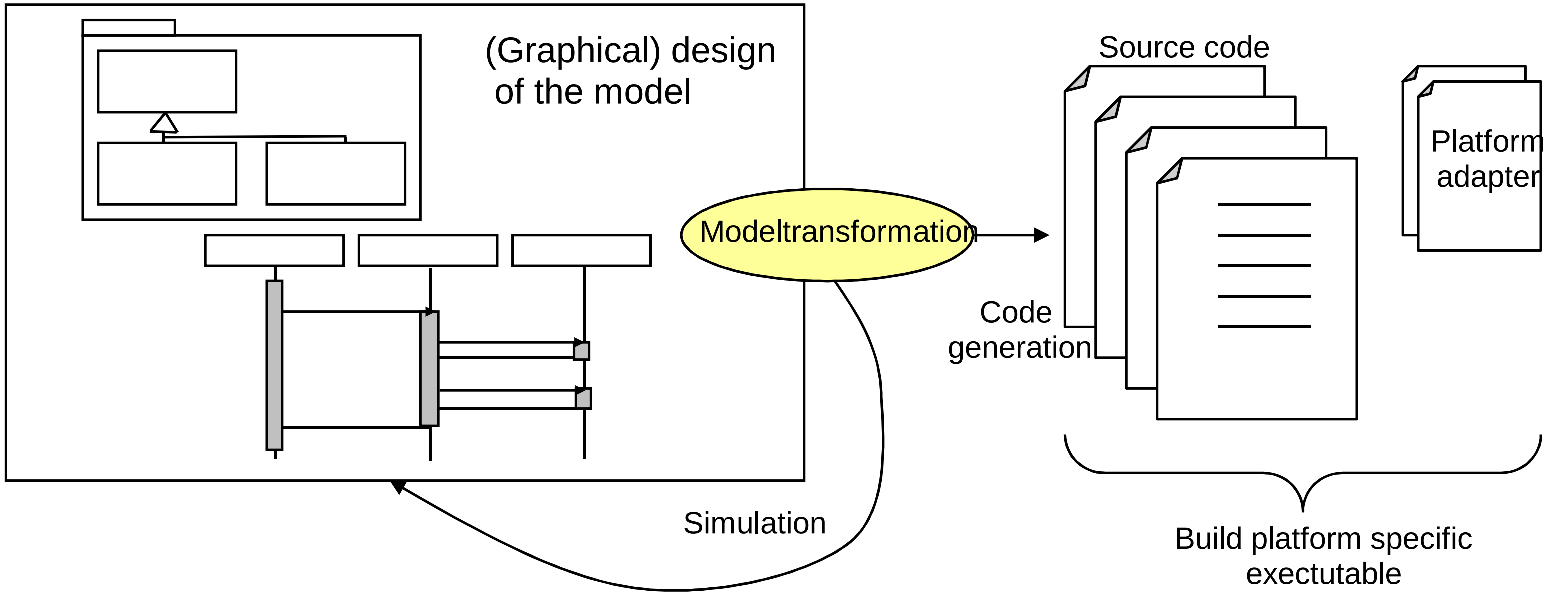
Give four quality aims which should be realized with high quality software?

Give the phases of the general top down model, explain shortly why it is also called a waterfall model and give the disadvantages!

Give the phases of the formalized software process and explain each with one sentence!

Modelling techniques: UML

Automated code generation and possibilities for high level simulation:



See: *http://casablanca.informatik.hu-berlin.de/database/MDA-UML/VL/V1-MODSOFT-I.1.Ueberblick.pdf*, Download 07.11.2006

Modelling techniques: aspect oriented programming

Model **cross cutting concerns** in object oriented solutions:

Additional functionality which is not immediatly relevant for the functionality of a software but very important for developement, error prevention, simulation and code investigation.

Very interesting for:

- Logging
- Pre and post conditioning
- Authentication
- Transaktion
- Memory control
- etc.

See: Ettl, Martin: Ein kurzer Überblick zur Aspektorientierten Programmierung (AOP). Internal Paper Wettzell 2006

Modelling techniques: aspect oriented programming

Aspects:

Aspects model the cross cutting cerncerns, by adding additional descriptions to the classes, which are compiled into the original source code.

So the original code with the functionality is not „contaminated“ by additional, for the wished functionality irrelevant code.

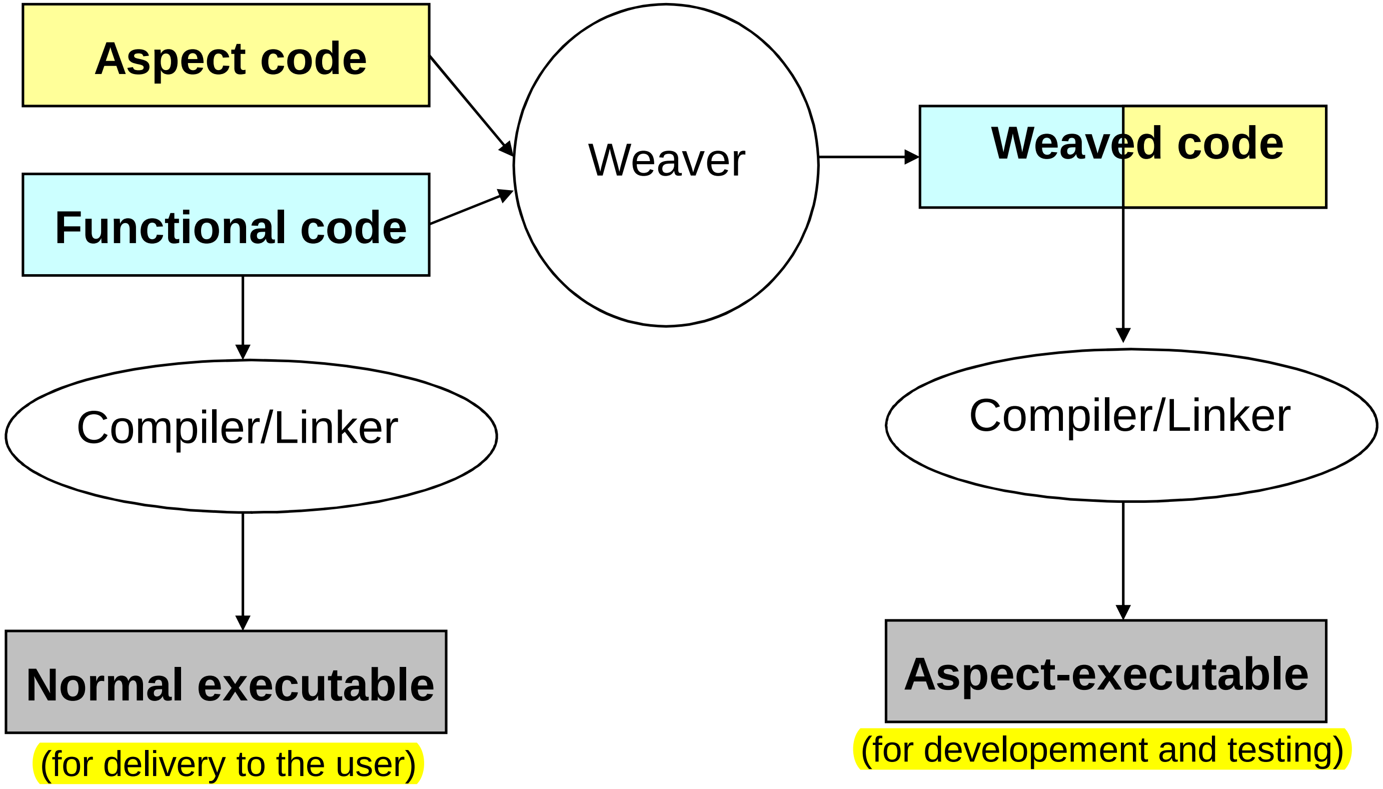
The aspects act at so called **join points**:

- Methode calls and returns
- Variable access
- Exception handling
- Initialization
- …

See: Ettl, Martin: Ein kurzer Überblick zur Aspektorientierten Programmierung (AOP). Internal Paper Wettzell 2006

Modelling techniques: aspect oriented programming

Mixing functional code with aspect code by a weaver (= pre-processor):



See: Ettl, Martin: Ein kurzer Überblick zur Aspektorientierten Programmierung (AOP). Internal Paper Wettzell 2006

Lecture 4: Software design

✓ Review: Software development method and process

✓ Modelling techniques

- Design/ programming rules
- Model driven architecture

Design Rules

Which problem is solved by the following unreadable C-code?

```
int v,i,j,k,l,s,a[99];
main ()
{
   for (scanf(„%d“,&s);*a-s;v=a[j*=v]-a[i],k=i<s,j+=(v=j<s&&(!k&&!!
   printf(2+“\n\%c“-(!1<<!j),“ #Q“[1^v?(1^j)&1:2])&&++1||a[i]<s&&v&&
   v-i+j&&v+i-j)) &&!(1%=s),v||(i==j?a[i+=k]=0:++a[i])>=s*k&&++a[--i]);
}
```

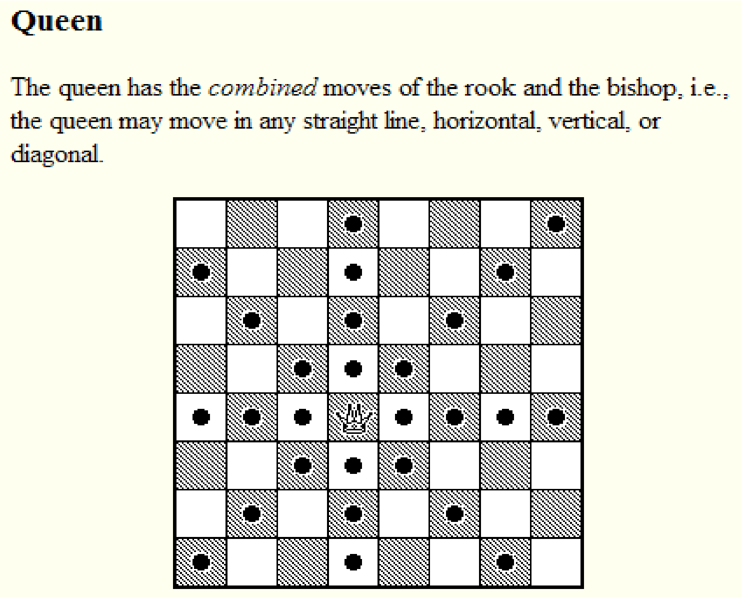
See: Schicker, E.: Script for the course „Programmierung C“, FH Regensburg ca. 1995

Design Rules

Which problem is solved by the following unreadable C-code?

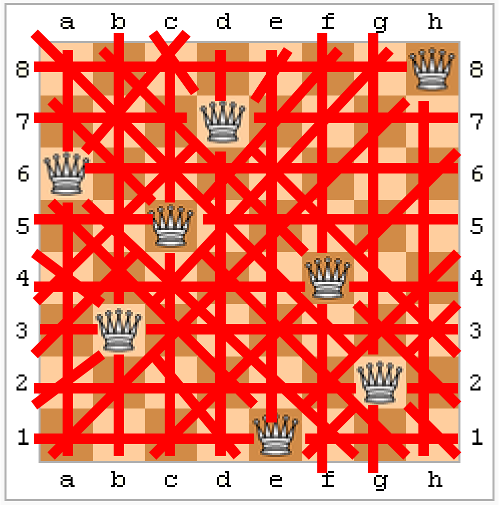
```
int v,i,j,k,l,s,a[99];
main ()
{
   for (scanf(„%d“,&s);*a-s;v=a[j*=v]-a[i],k=i<s,j+=(v=j<s&&(!k&&!!
   printf(2+“\n\%c“-(!1<<!j),“ #Q“[1^v?(1^j)&1:2])&&++1||a[i]<s&&v&&
   v-i+j&&v+i-j)) &&!(1%=s),v||(i==j?a[i+=k]=0:++a[i])>=s*k&&++a[--i]);
}
```

Answer: Find all solutions for n queens on a n x n chess board so that no queen is in danger because of the others.



Queen

The queen has the *combined* moves of the rook and the bishop, i.e., the queen may move in any straight line, horizontal, vertical, or diagonal.



One possible solution

See: Schicker, E.: Script for the course „Programmierung C“, FH Regensburg ca. 1995
See: http://www.chessvariants.com/d.chess/chess.html, Download 14.12.2008
See: http://de.wikipedia.org/wiki/Damenproblem, Download 14.12.2008

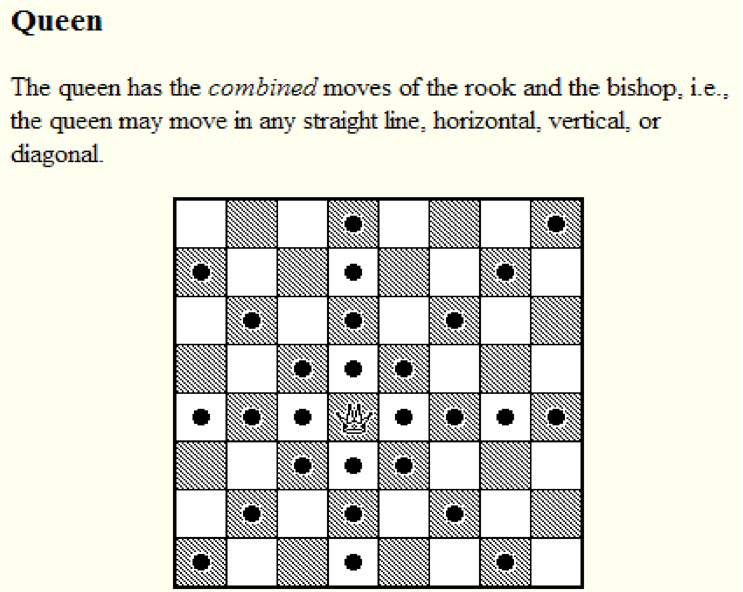
Design Rules

Which problem is solved by the following unreadable C-code?

```
int v,i,j,k,l,s,a[99];
main ()
{
   for (scanf(„%d“,&s);*a-s;v=a[j*=v]-a[i],k=i<s,j+=(v=j<s&&(!k&&!!
   printf(2+“\n\%c“-(!1<<!j),“ #Q“[1^v?(1^j)&1:2])&&++1||a[i]<s&&v&&
   v-i+j&&v+i-j)) &&!(1%=s),v||(i==j?a[i+=k]=0:++a[i])>=s*k&&++a[--i]);
}
```

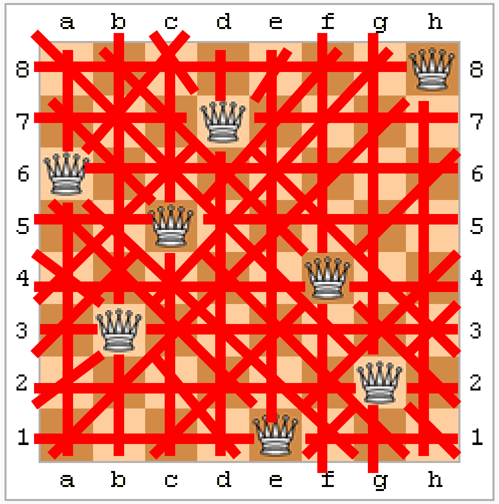
Not well!!!

Answer: Find all solutions for n queens on a n x n chess board so that no queen is in danger because of the others.



Queen

The queen has the *combined* moves of the rook and the bishop, i.e., the queen may move in any straight line, horizontal, vertical, or diagonal.



One possible solution

See: Schicker, E.: Script for the course „Programmierung C“, FH Regensburg ca. 1995
See: http://www.chessvariants.com/d.chess/chess.html, Download 14.12.2008
See: http://de.wikipedia.org/wiki/Damenproblem, Download 14.12.2008

Design Rules

Design rules contain (e.g. C/C++ Design Rules at the Geodetic Observatory Wettzell):

- Guidelines for the project design (language, programming language, software developement process, UML)

- Guidelines for the source code generation (structure, notation, special guidlines, helping hints, design and code patterns => strategies for development)

- Guidelines for the executable build

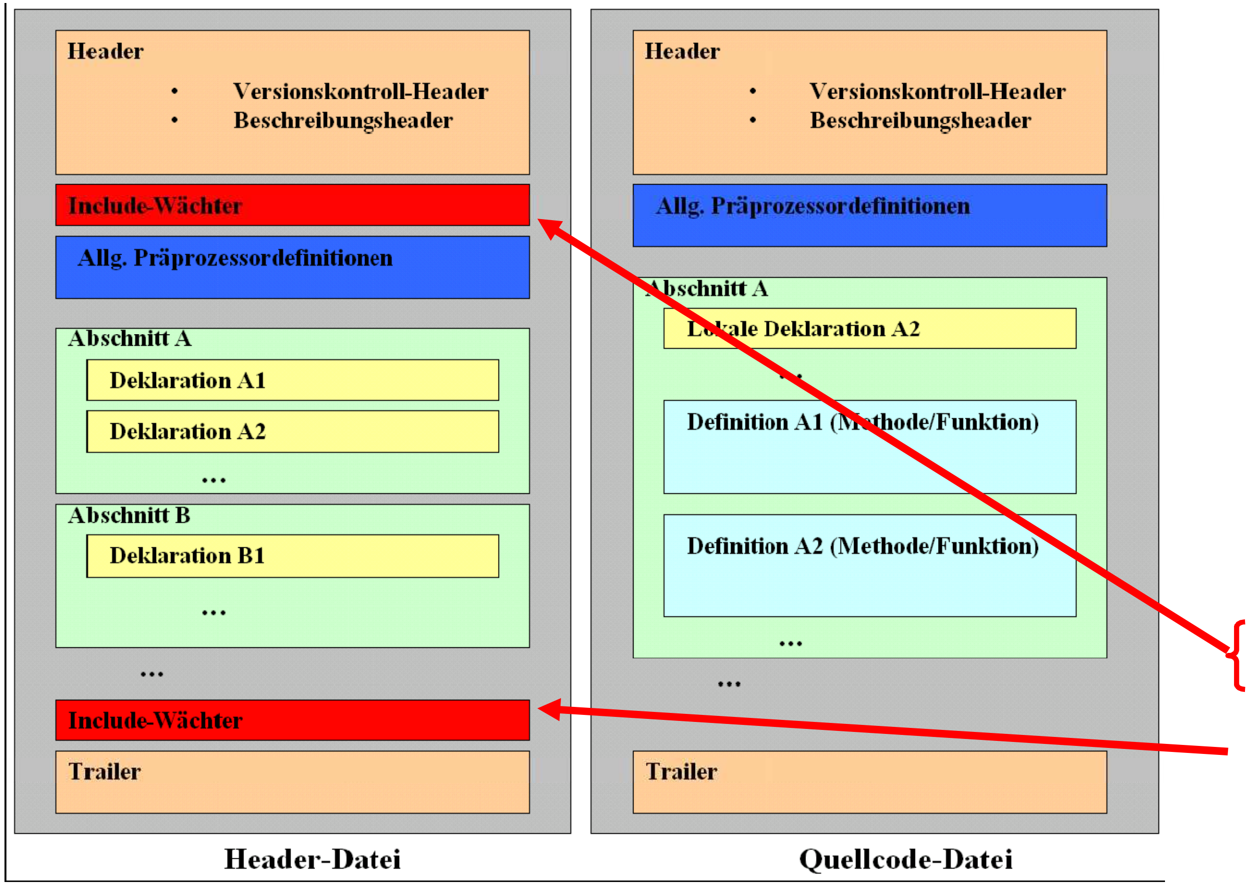
- Guidelines for testing

- Guidelines for documentation

(Refernce: Dassing, Reiner; Lauber, Pierre; Neidhardt, Alexander: Design-Rules für die objektorientierte Programmierung in C++ und die strukturierte Programmierung in C, Fundamentalstation Wettzell 2004)

Design Rules – Source code generation

e.g. the code file structure:



```
With „Include-Guardian“
in .h-files
#ifndef __FILEIO__
#define __FILEIO__
... // Headerdeclarations
#endif //__FILEIO__
```

(Refernce: Dassing, Reiner; Lauber, Pierre; Neidhardt, Alexander: Design-Rules für die objektorientierte Programmierung in C++ und die strukturierte Programmierung in C, Fundamentalstation Wettzell 2004)

Design Rules – Source code generation

e.g. the extended Hungarian Notation of Charles Simonya:

Notation extension 4: access (global, private, ...)

+ Notation extension 3: handling (static, constant, ...)

+ Notation extension 2: combination (array, pointer,...)

+ Notation extension 1: type (int, char, ...)

+ Understandable classifier

<u>For example</u>

```
int iCheckErrorControl;
char azcFirstName[25];
void vGRD2JD (short sYear, short sMonth, short sDay, short sHour,
                           short sMinute, short sSecond, double *pdJD);
```

(Refernce: Dassing, Reiner; Lauber, Pierre; Neidhardt, Alexander: Design-Rules für die objektorientierte Programmierung in C++ und die strukturierte Programmierung in C, Fundamentalstation Wettzell 2004)

Design Rules – Source code generation

e.g. programming basics C:

- Do not use compiler directives (macros)

- Do not use inline code (only when timing relevant functionality)

- All variables have to be initialized

- Avoid the usage of dynamic memory (heap)

- Goto is not allowed (only in special cases)

- Time optimization should be done immediatly and not after the first tests

- Implicite type casting is not permitted

- Boolean return values are not used. Better to use integer values with detailed errorcodes.

- Foraign languages or modules have to be encapsulated.

- Use the standard libraries than using own functionality.

- In most cases it is good to use software packages if necessary instead of writing own code.

(Refernce: Dassing, Reiner; Lauber, Pierre; Neidhardt, Alexander: Design-Rules für die objektorientierte Programmierung in C++ und die strukturierte Programmierung in C, Fundamentalstation Wettzell 2004)

Design Rules – Source code generation

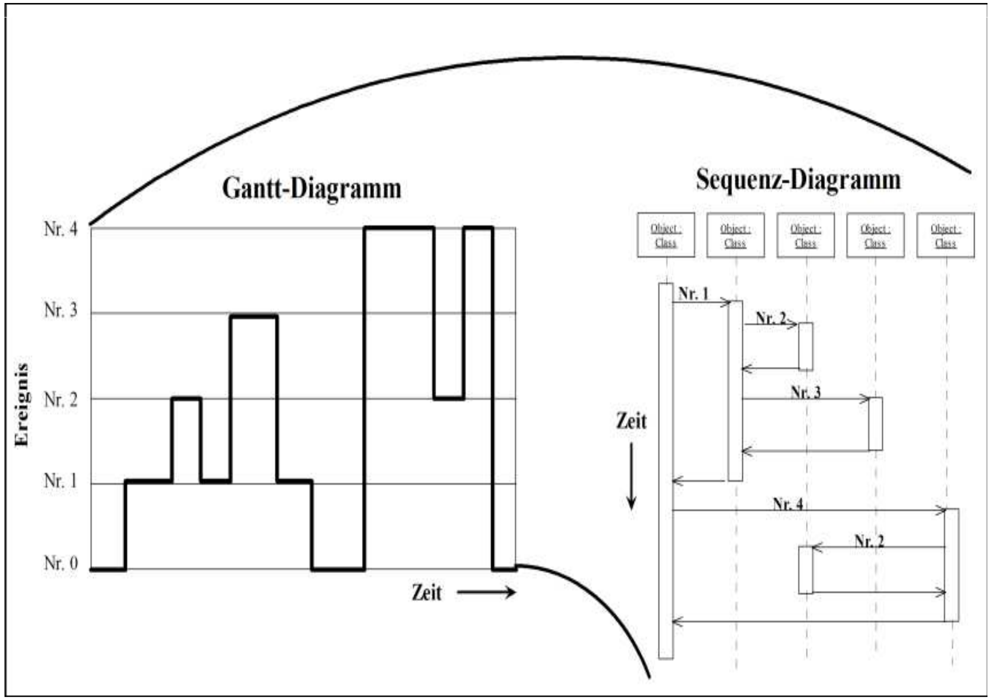
e.g. logging („log book“):

Each message output which doesn‘t belong to a user interaction is a log and is categorized with:

- ALERT: System must stop
- ERROR: System detected wrong situation, which can be handled
- EVENT: A system state is reached
- DEBUG: A detailed info about the internal functionality during run time

An additional option is

- TRACE: Write a path through the functionality calls, which is time based sequence diagram



(Refernce: Dassing, Reiner; Lauber, Pierre; Neidhardt, Alexander: Design-Rules für die objektorientierte Programmierung in C++ und die strukturierte Programmierung in C, Fundamentalstation Wettzell 2004)

Design Rules – Source code generation

e.g. comments

- Dedicated comments can be interpreted by the automatical documentation tool Doxygen
- Code file starts with a concurrent version control (CVS) header and ends with a trailer
- Section comments
- Function and methode header

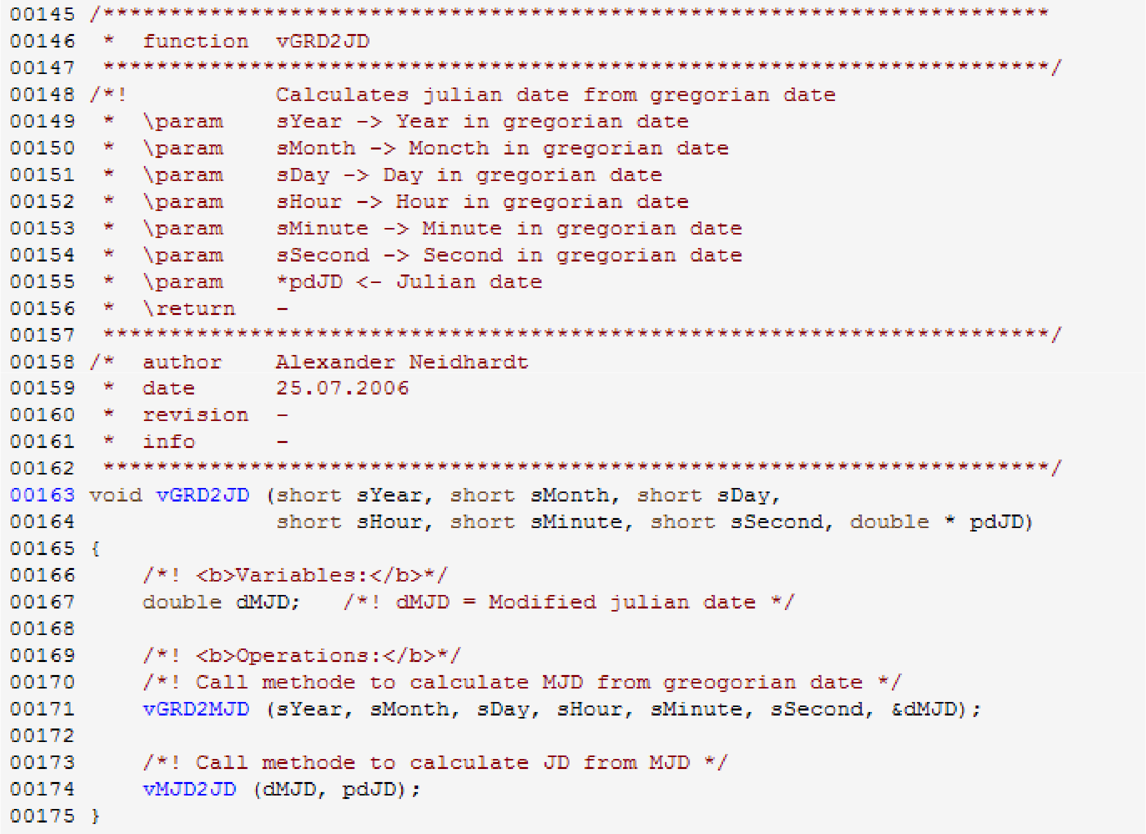
```
/***************************************************************
* CVS Concurrent Versions Control
* -------------------------------------------------------------
* $RCSfile: $
* $Revision: $
* -------------------------------------------------------------
* $Author: $
* $Date: $
* $Locker: $
* -------------------------------------------------------------
* $Log: $
*
***************************************************************/
/*! \file
* \brief [Kurzbeschreibung] \n
*
* [Beschreibung]
***************************************************************
* Defined precompiler definitions: [Externe Definitionen] <br>
* Defined namespaces: [Nutzbare Namensräume] <br>
* Defined exeption-handles: [Ausnahmebehandlungen] <br>
* ...
***************************************************************/
```

```
/************************************************
* function [Funktionsname]
************************************************
/*! [Funktionsbeschreibung]
* \param [Parameter1] -> [Parameterbeschreibung]
* \param [Parameter2] -> [Parameterbeschreibung]
* ...
* \return <- [Returnangaben]
*************************************************************/
/* author [Name][Name]...
* date [Datum]
* revision [Revisionsnummer]
* [Beschreibung der Revision]
* info [Wichtige Zusatzinformation]
*************************************************************/
```

(Refernce: Dassing, Reiner; Lauber, Pierre; Neidhardt, Alexander: Design-Rules für die objektorientierte Programmierung in C++ und die strukturierte Programmierung in C, Fundamentalstation Wettzell 2004)

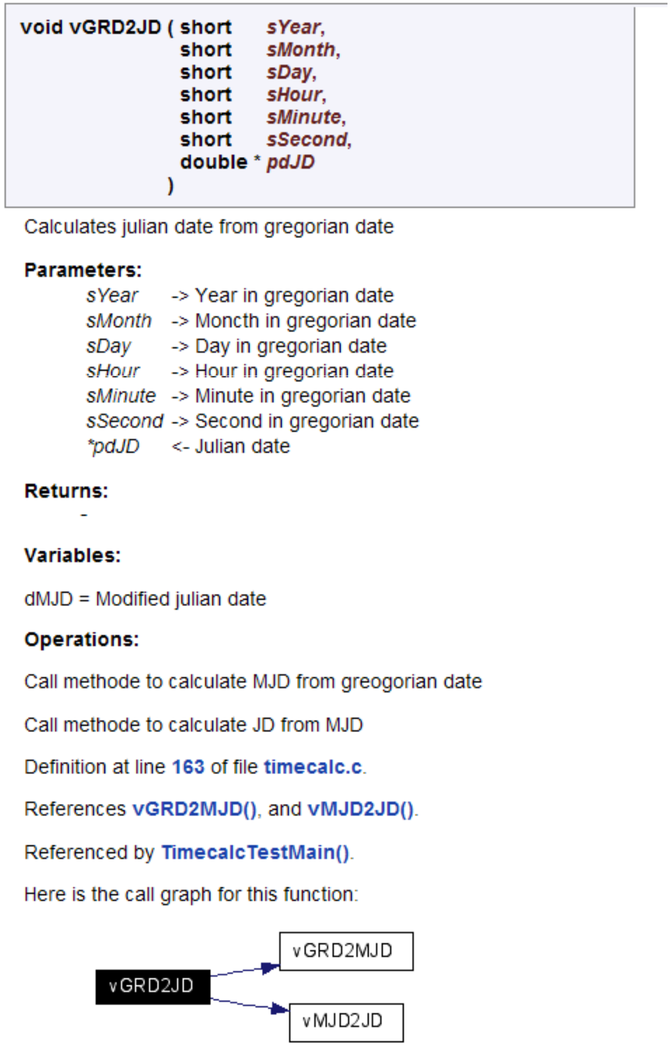
Design Rules – Generated documentation (doxygen)

Example Code:



```
00145 /***********************************************************************
00146  *  function  vGRD2JD
00147  ***********************************************************************/
00148 /*!           Calculates julian date from gregorian date
00149  *  \param    sYear -> Year in gregorian date
00150  *  \param    sMonth -> Moncth in gregorian date
00151  *  \param    sDay -> Day in gregorian date
00152  *  \param    sHour -> Hour in gregorian date
00153  *  \param    sMinute -> Minute in gregorian date
00154  *  \param    sSecond -> Second in gregorian date
00155  *  \param    *pdJD <- Julian date
00156  *  \return   -
00157  ***********************************************************************/
00158 /*  author    Alexander Neidhardt
00159  *  date      25.07.2006
00160  *  revision  -
00161  *  info      -
00162  ***********************************************************************/
00163 void vGRD2JD (short sYear, short sMonth, short sDay,
00164              short sHour, short sMinute, short sSecond, double * pdJD)
00165 {
00166     /*! <b>Variables:</b>*/
00167     double dMJD;   /*! dMJD = Modified julian date */
00168
00169     /*! <b>Operations:</b>*/
00170     /*! Call methode to calculate MJD from greogorian date */
00171     vGRD2MJD (sYear, sMonth, sDay, sHour, sMinute, sSecond, &dMJD);
00172
00173     /*! Call methode to calculate JD from MJD */
00174     vMJD2JD (dMJD, pdJD);
00175 }
```

Example documentation:



```
void vGRD2JD ( short    sYear,
               short    sMonth,
               short    sDay,
               short    sHour,
               short    sMinute,
               short    sSecond,
               double * pdJD
             )
```

Calculates julian date from gregorian date

Parameters:

sYear	-> Year in gregorian date
sMonth	-> Moncth in gregorian date
sDay	-> Day in gregorian date
sHour	-> Hour in gregorian date
sMinute	-> Minute in gregorian date
sSecond	-> Second in gregorian date
**pdJD*	<- Julian date

Returns:

-

Variables:

dMJD = Modified julian date

Operations:

Call methode to calculate MJD from greogorian date

Call methode to calculate JD from MJD

Definition at line **163** of file **timecalc.c**.

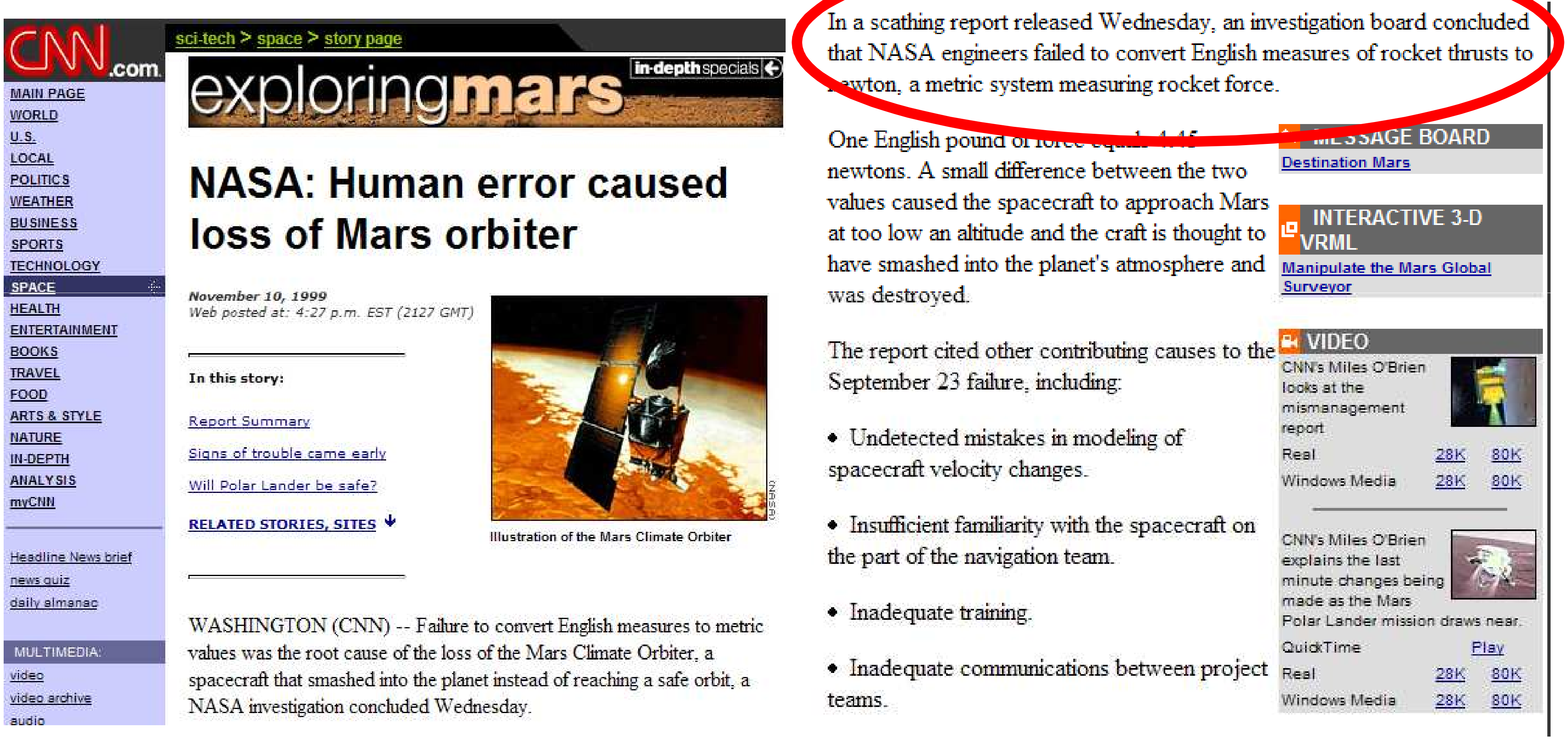
References **vGRD2MJD()**, and **vMJD2JD()**.

Referenced by **TimecalcTestMain()**.

Here is the call graph for this function:

Design Rules – design failures

Disasterous design errors: Mars orbiter (1999)



NASA: Human error caused loss of Mars orbiter

November 10, 1999
Web posted at: 4:27 p.m. EST (2127 GMT)

In this story:

Report Summary

Signs of trouble came early

Will Polar Lander be safe?

RELATED STORIES, SITES

Illustration of the Mars Climate Orbiter

WASHINGTON (CNN) -- Failure to convert English measures to metric values was the root cause of the loss of the Mars Climate Orbiter, a spacecraft that smashed into the planet instead of reaching a safe orbit, a NASA investigation concluded Wednesday.

In a scathing report released Wednesday, an investigation board concluded that NASA engineers failed to convert English measures of rocket thrusts to newton, a metric system measuring rocket force.

One English pound of force equals 4.45 newtons. A small difference between the two values caused the spacecraft to approach Mars at too low an altitude and the craft is thought to have smashed into the planet's atmosphere and was destroyed.

The report cited other contributing causes to the September 23 failure, including:

- Undetected mistakes in modeling of spacecraft velocity changes.
- Insufficient familiarity with the spacecraft on the part of the navigation team.
- Inadequate training.
- Inadequate communications between project teams.

CNN's Miles O'Brien looks at the mismanagement report

CNN's Miles O'Brien explains the last minute changes being made as the Mars Polar Lander mission draws near.

For more Software Horror Stories see homepage of Nachum Dershowitz, Tel Aviv University (http://www.cs.tau.ac.il/~nachumd/horror.html)

See: http://www4.cnn.com/TECH/space/9911/10/orbiter.03/, Download 17.12.2006

Lecture 4: Software design

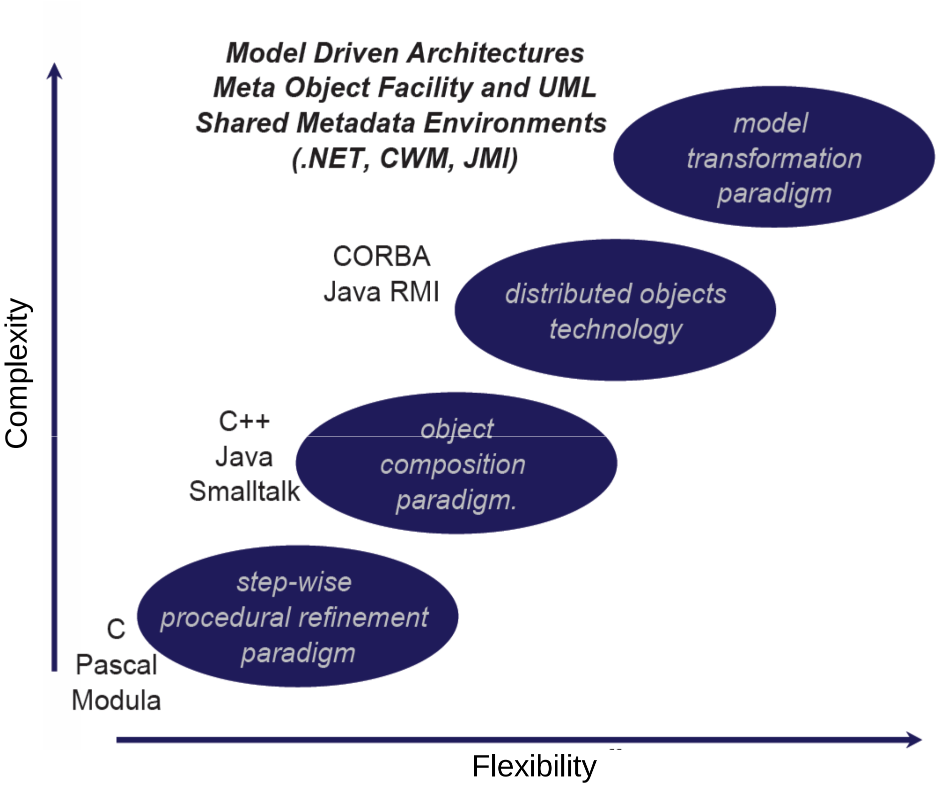
✓ Review: Software development method and process

✓ Modelling techniques

✓ - Design/ programming rules

- Model driven architecture

Modelling techniques: model driven architecture


Model Driven Architectures
Meta Object Facility and UML
Shared Metadata Environments
(.NET, CWM, JMI)
model transformation paradigm
CORBA
Java RMI
distributed objects technology
C++
Java
Smalltalk
object composition paradigm.
C
Pascal
Modula
step-wise procedural refinement paradigm
Complexity
Flexibility

The change of software construction paradigm

Advantages:

- Reliability/Robustness/Funcionality/ Correctness/Testability: Simulation, Design rules
- Portability/Scalability: Platform, producer and computer language independent design
- Maintainability: Low cost and fast release management, bug fix management
- Compactness/Re-usability/Modularity: Usage of model repositories
- Comprehensability/Understandability: immediate documentation with diagrams
- Schedulability/Efficiency/Flexibility: high level design with automated code generation

→ **A better way to realize quality aims**

See: *http://casablanca.informatik.hu-berlin.de/database/MDA-UML/VL/V1-MODSOFT-I.1.Ueberblick.pdf*, Download 07.11.2006

Thank you