

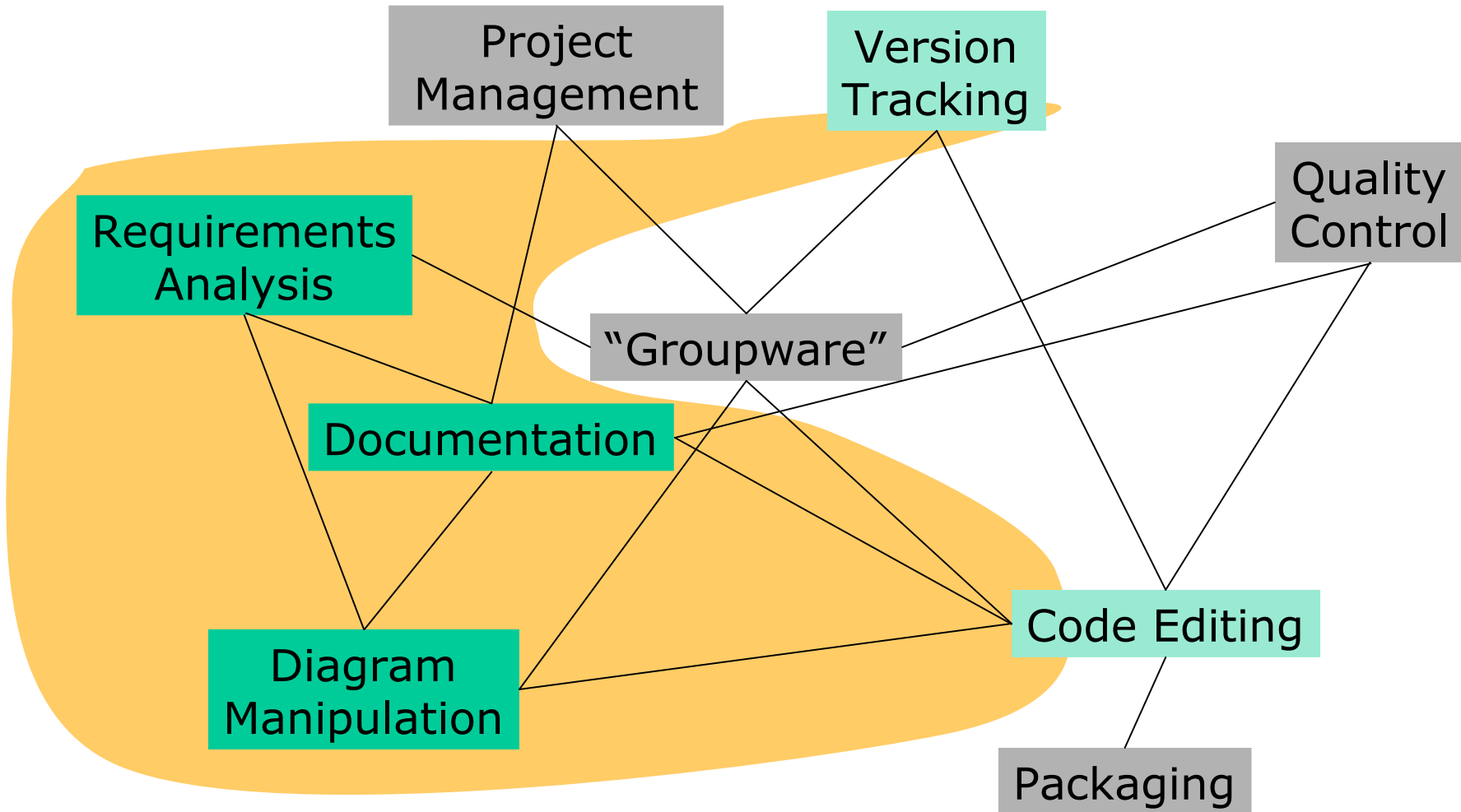
# Model-centric Tools

---

Model-centric Tools

# Reminder: Model-Centric Tool

---



# Model-centric Tools

---

## Requirements management tools

- ❑ use of generic tools for requirements capturing and analysis
- ❑ requirements management tools

## Diagrammatic tools

- ❑ creation and representation of high level models through diagrams
- ❑ transitions based on a software model

## Also covered here: project management tools

- ❑ project planning and controlling
- ❑ tools for project status visualization and overview
- ❑ ..

# Requirements Engineering (1)

---

*Generic tools:*

- text processing (descriptions, contracts, glossaries)
- graphics (see, for example, use case diagrams)

Problems with text and graphic tools:

- relating analysis documents, e.g.,
  - use cases diagrams visualizing textual descriptions and
  - glossaries explaining entities found on diagrams
- supporting traceability
- representing non-functional requirements for implementation decisions (interfaces, queries, ...)
- specifying test cases from requirement descriptions
- capturing relationships between phases

# Requirements Engineering (2)

---

## *Specific* requirements engineering tools

- Emphasis on documents
  - managing requirements through a set of individual documents
  - example: IBM Rational RequisitePro
- Emphasis on relationships
  - defining relationships between analysis documents
  - example: Telelogic DOOR

# IBM Rational RequisitePro

---

A requirements management tool for the communication of project goals and collaborative development in project teams.

Properties:

- ❑ integration with Microsoft Word
- ❑ database with Word document synchronization
- ❑ definable requirement, attribute, document types; queries and filters.
- ❑ notification about changes of requirements
- ❑ traceability views for parent/child relationships
- ❑ integration with IBM software development tools

# Example

reflect the evolving business needs. A suspect link (red slashed arrow in Figure 9) indicates that use case UC1.2 may need to be revisited due to a change in business need BUS1.4. Querying on suspect links answers the last question presented at the beginning of the article: *Are my use cases staying in touch with the evolving business needs they are supposed to solve?*

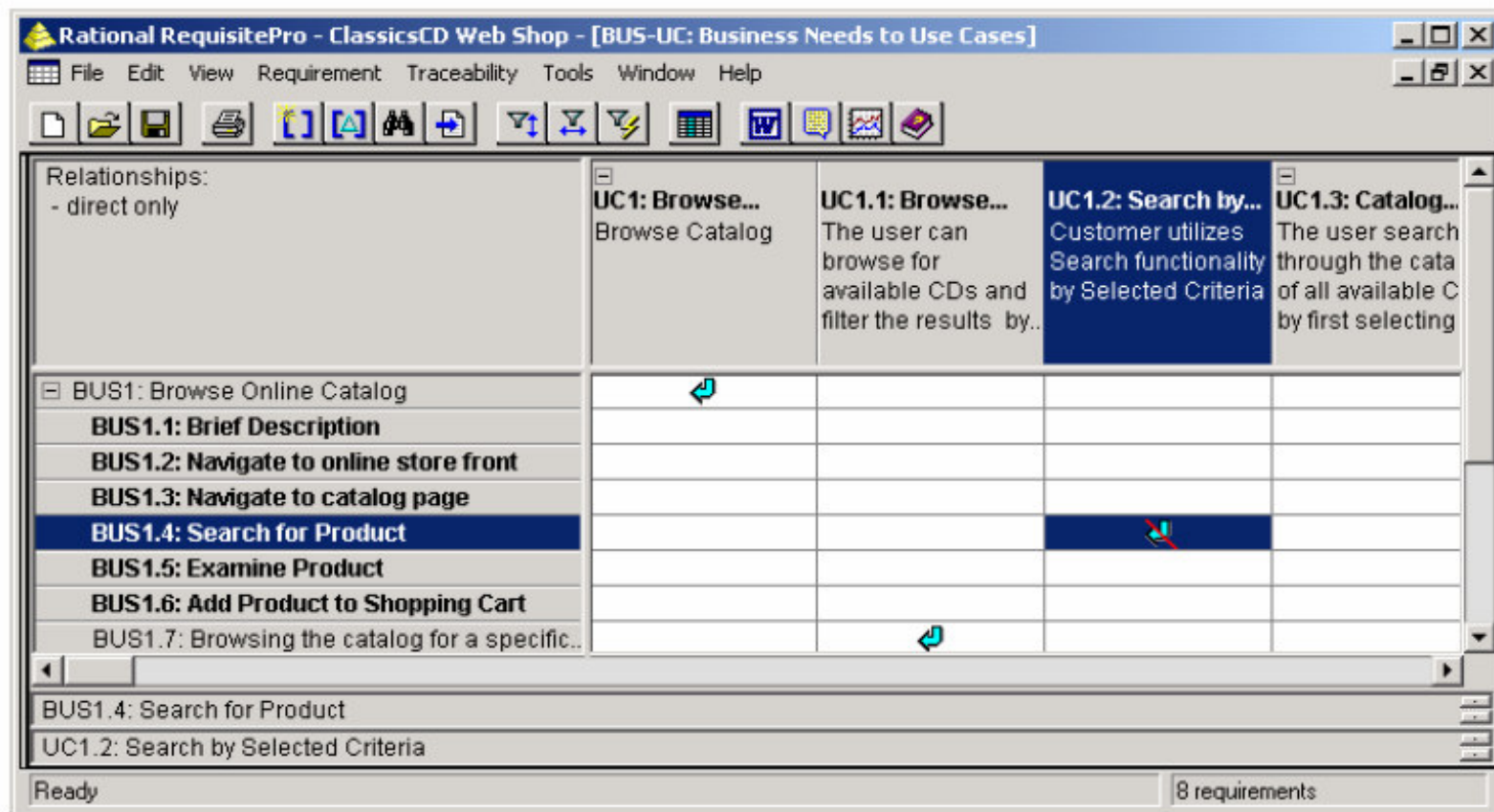


Figure 9: Traceability relationships between use cases and business needs

# Telelogic DOOR

---

Captures, traces, and manages information to support compliance with specified requirements

Properties:

- insertion of data from various sources into documents
- multi-user environment (editable documents, sections)
- traceability over user-defined relationships, e.g.,
  - requirements to design
  - design to code
  - requirements for test
  - requirements for tasks
- generation of link reports across as many levels as required; link display in the same view
- customizable views

# Telelogic DOOR (2)

Formal module '/Sports utility vehicle 4x2/Requirements/User Requirements' current 2.1 (1998) - DOORS

File Edit View Insert Link Analysis Table Tools User DOORSrequireIT Help

F - Budget All levels

Obj id	User requirements for SUV 4x2	Spent	Remaining	Verification Method	Risk	Last Modified On
SOW 37	<b>3.1.4 Fuel economy</b>	0	146	No Verification Needed		11 April 1997
SOW 38	Users shall be able to obtain fuel consumption better than that provided by the 95% of cars built in 1996.	0	67		High	27 March 1997
SOW 39	Users shall be able to accelerate from 0 to 100 Kilometers per hour in 10 seconds.	0	79		Medium	03 December 1997
SOW 364	Users shall be able to accelerate from 0 to 100 Kilometers per hour in 8 seconds.	0	79		High	03 December 1997
SOW 40	<b>3.1.5 Safety</b>	0	20			11 April 1997
		0	0	Demonstration	Medium	03 December 1997
		0	20	Demonstration	Medium	26 December 1997
		0	95			11 February 1997
		0	81			11 April 1997
		0	81	Analysis	Low	27 March 1997
		0	14		High	
		0	14		Medium	1997
		0	475		Low	11 April 1997
		0	475			11 February 1997

/DOORS Database/Software Projects/DOORS Family/DOORS/DOORS Black - DOORS

File Edit View Tools Help

DOORS Database

- Company Standards
- Management Projects
- Services Projects
- Software Projects
  - DOORS Family
    - DOORS
      - DOORS Black
        - CPS sub-project
        - Design
        - Experimental Designs
        - Requirements
        - Software
        - Test
        - Use Cases
        - Meeting Minutes
        - Product Plan
        - Project Plans
      - DOORS Blue
      - DOORS Navy
      - DOORS White

Name	Type	Description
CPS sub-project	Project	Change Proposal System
GUI Redesign sub-project	Project	Full redesign of GUI
Design	Folder	All design documentation
Experimental Designs	Folder	Designs used to test new ideas
Requirements	Folder	All requirements documents
Software	Folder	Code data for detailed traceability
Test	Folder	All test specifications
Use Cases	Folder	Traced to Requirements
Meeting Minutes	Formal	Weekly meetings
Product Plan	Formal	Product Plan
Project Plans	Formal	Development Project Plans

Username: Administrator

# Diagrammatic Tools

---

Based on a visual language; most often UML

Levels of support

- free drawing
- shape-based drawing
- shape-based drawing with (semantic) rules
- modeling tools
  - manipulation of *one* underlying software model through *different* graphical representations
  - thus, diagrams are interrelated
  - model amendments (e.g., application of patterns)
  - code generation and reengineering

# Basic Functionality of Diagrammatic Tools

---

Create and develop diagrams ...

- ❑ drawing capabilities of traditional drawing tools combined with
- ❑ diagram manipulation techniques that facilitate creation of diagrams in your language (e.g. UML)

... which visualize results of different phases

- ❑ requirements analysis
- ❑ design phase

(Non-) Interoperability

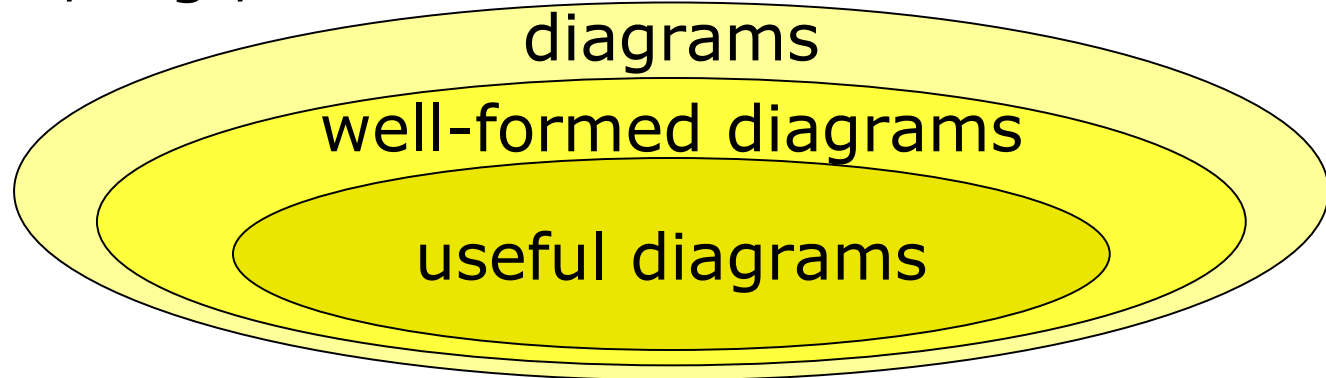
- ❑ Created diagrams need to be in a format that is understood by other tools (XMI)
- ❑ This is not a “nice-to-have” feature but essential to enable transitions between phases.

# Model Verification

---

Modeling tools usually offer limited support for model checking

- ❑ mainly covering language syntax
- ❑ semantics are checked to a limited extent
- ❑ based on, e.g., UML meta model

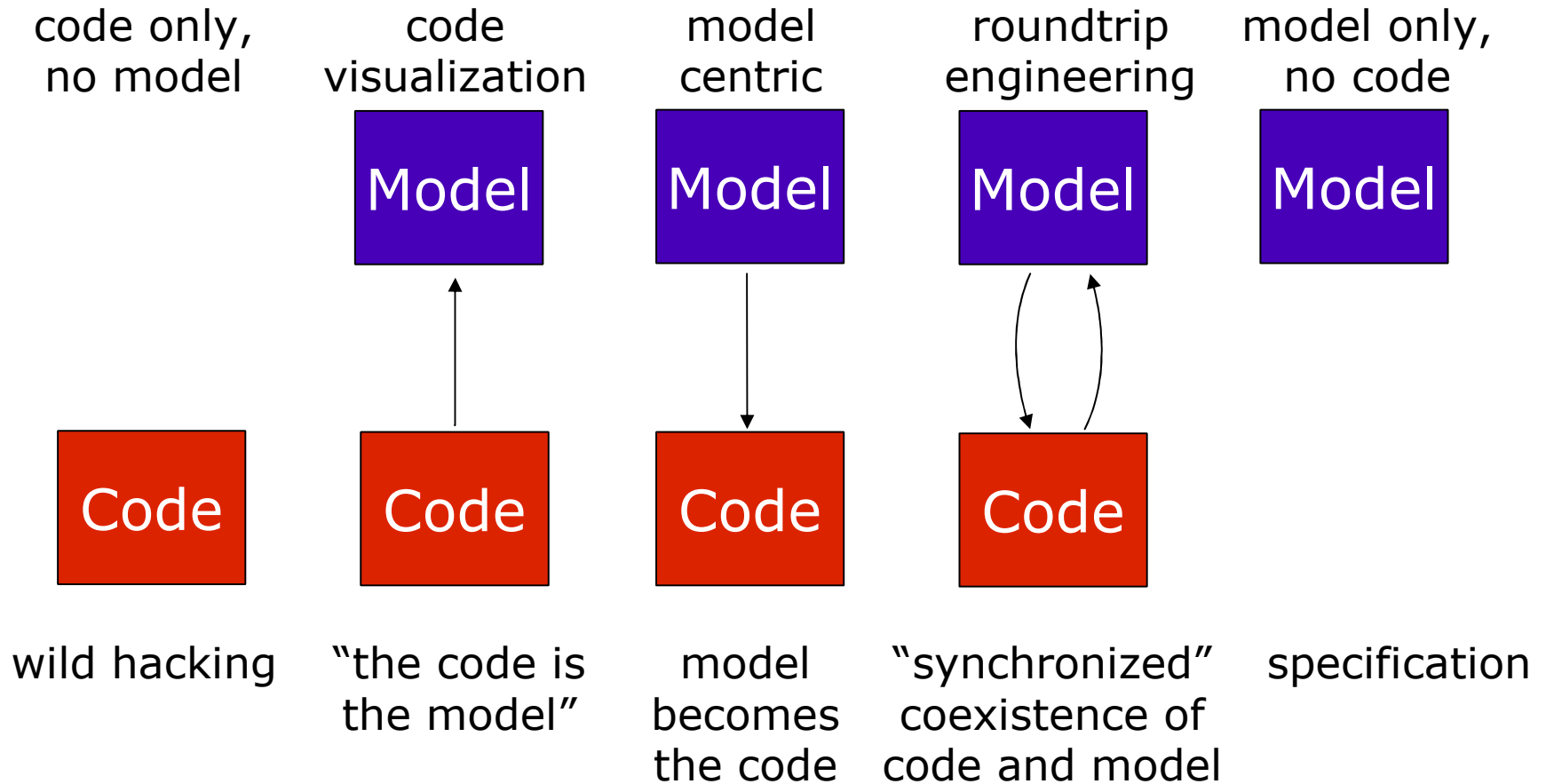


Reasons:

- ❑ Languages like the UML have weak semantics.
- ❑ The intention of models cannot be checked automatically.

# Model - Code Synchronization

---



# Transitions: Diagram Relationships

---

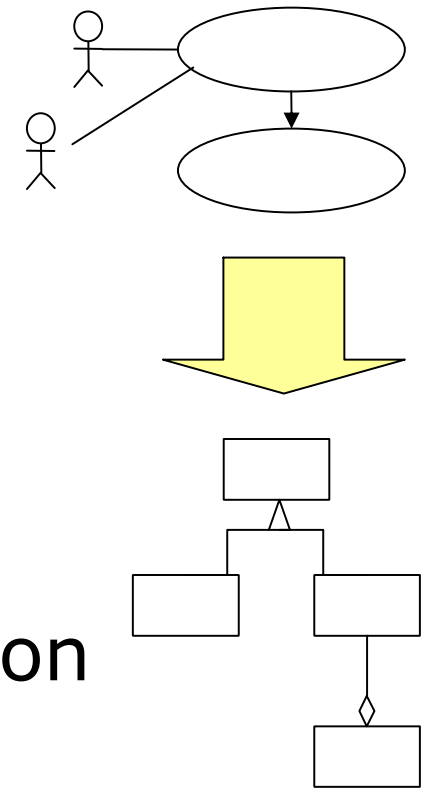
At the end of each development phase, you transform your artifacts to suit the next phase.

This can be done:

- ❑ manually
- ❑ semi-automatically

Model-centric tools usually offer some support for this.

Keep in mind that a full transformation is not possible unless you specify *all* implementational details in the model.



# Transition: Semi-automatic Code Generation

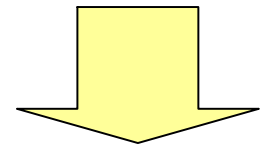
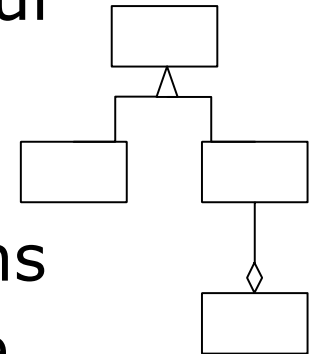
Example of a semi-automatic transition: Many UML tools can generate skeleton code based on your diagrams

Generation of partial code based on model:

- simplest case: class diagram + annotations
- *Together*: diagrams in lock-step with code (missing semantics of code are expressed as special comments)
- perspective: diagrams + constraints  $\Rightarrow$  MDA

Observation: impedance mismatch between the artifacts of the two phases

- this makes roundtripping difficult



```
class Something
  extends STEise
{
  public void doIt ()
  {
    ich (kann, gar)
    nicht sehen,
    was :
    ich = tippe :
    blauer elefant
    ? weiße() Füße
    : grüne_Füße()
    soße(grün) :
    switch(lecker) {
    ja:
    essen();doIt();
    nein: break;
    }
  }
}
```

# Model-Driven Architecture

---

Models have well-defined notation and semantics

- cornerstone to understanding large systems.

System building is organized around the models

- series of transformations between models,
- organized into an architectural framework of layers and transformations.

Formal metamodels

- facilitates meaningful integration and transformation among models,
- are the basis for automation through tools.

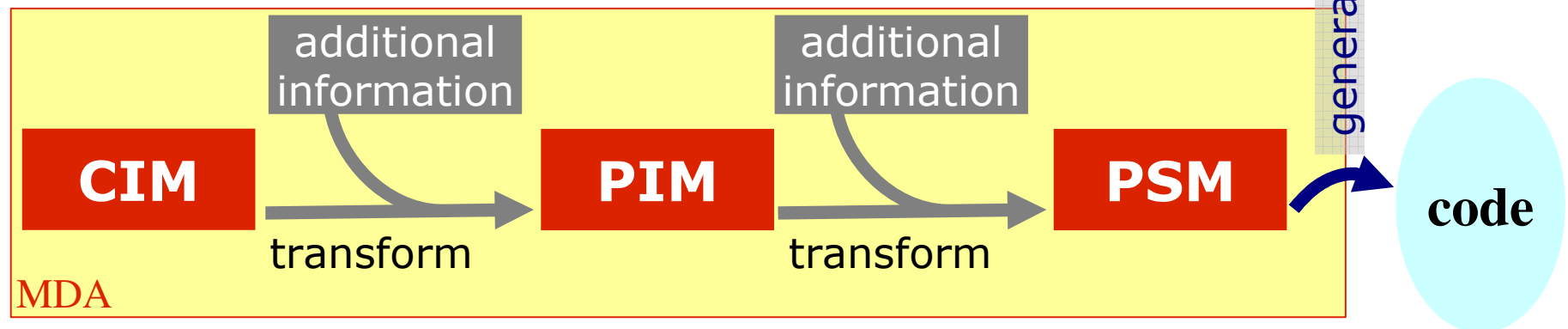
Acceptance will require cross-vendor openness

- Exchanging models between tools is vital.

# Model-Driven Architecture: Methodology

Three related software models:

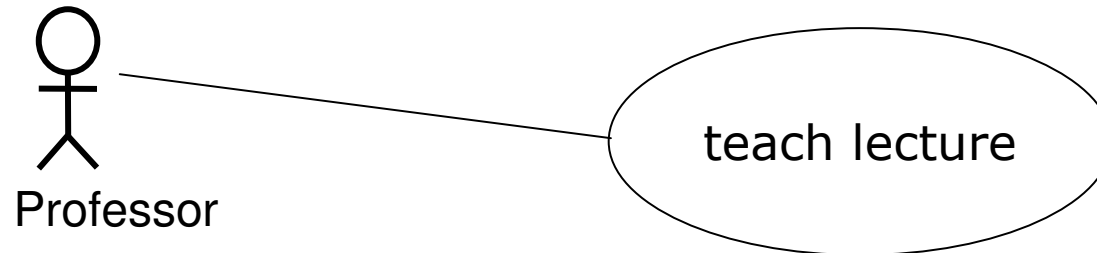
- ❑ *Computation*-Independent Model (CIM): domain-specific that is expressed in the vocabulary of the domain practitioner
- ❑ *Platform*-Independent Model (PIM): Often targets a platform-neutral virtual machine (composed of specific objects and services)
- ❑ *Platform-Specific* Model (PSM): specifies how the PIM will be mapped to the target platform



# MDA Example: CIM

---

Excerpt from a domain specific model:



- There are lectures.
- Professors teach lectures.
- Professors have names.

Comment: UML is probably not used by domain experts.

# MDA Example: CIM (2)

---

For an example we use the **Concept-oriented Content Management** platform developed at STS

- ❑ based on the concept of assets
- ❑ further details skipped, this is only an example

Sample CIM in Asset Definition Language (ADL):

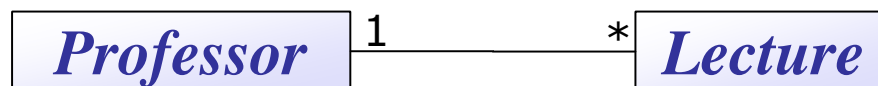
```
cim University
class Professor {
    concept
        characteristic name
        relationship lectures :Lecture*
}
class Lecture
```

# MDA Example: PIM

---

Platform independent model:

- ❑ the names of professors are implemented by string objects,
- ❑ the names are unique,
- ❑ lectures have a title given by a string object,
- ❑ lectures are taught by exactly one professor, and
- ❑ there should be a navigational path from a lecture to the teaching professor.



# MDA Example: PIM (2)

---

In ADL:

```
pim University
from ComputationalEntities import String
class Professor {
  concept
    characteristic name :String
    relationship lectures :Lecture*
      ; relationship "lectures" can be inherited from cim
    constraint uniqueName {
      let p :Professor, p = self or p.name # name
    }
}

class Lecture {
  concept
    characteristic title :String
    relationship teacher :Professor
    constraint uniqueTeacher {
      let p1 :Professor, let p2 :Professor,
      p1 = p2 or
      not ({self}<=p1.lectures and {self}<=p2.lectures)
    }
}
```



computational  
domain

# MDA Example: PSM

---

The platform independent model can then be transformed (automatically) into a model tailored to a specific platform.

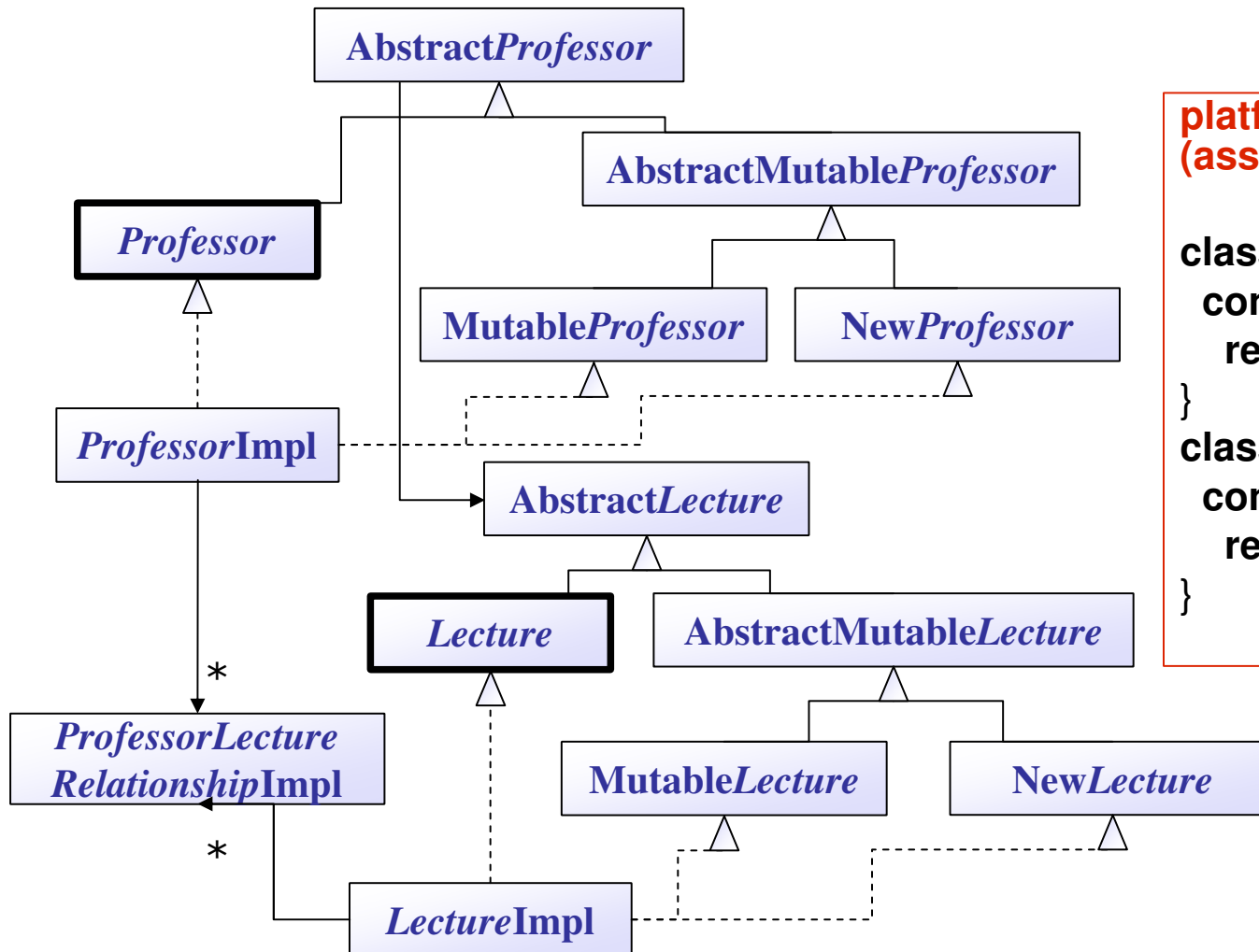
To Java in ADL:

```
psm[Java] University
from Java import java.lang.String
from Java-API[university] import
                                university.LectureIterator,
                                university.AbstractProfessor

class Professor {
    concept
        characteristic name      :java.lang.String
        relationship  lectures  :LectureIterator
}

class Lecture {
    concept
        characteristic title     :java.lang.String
        relationship  teacher   :AbstractProfessor
}
```

# MDA Example: PSM (2)



**platform dependent model (assets):**

```

class Professor {
  concept
  relationship l :Lecture*
}
class Lecture {
  concept
  relationship t :Professor
}
  
```

# MDA Example: PSM (3)

---

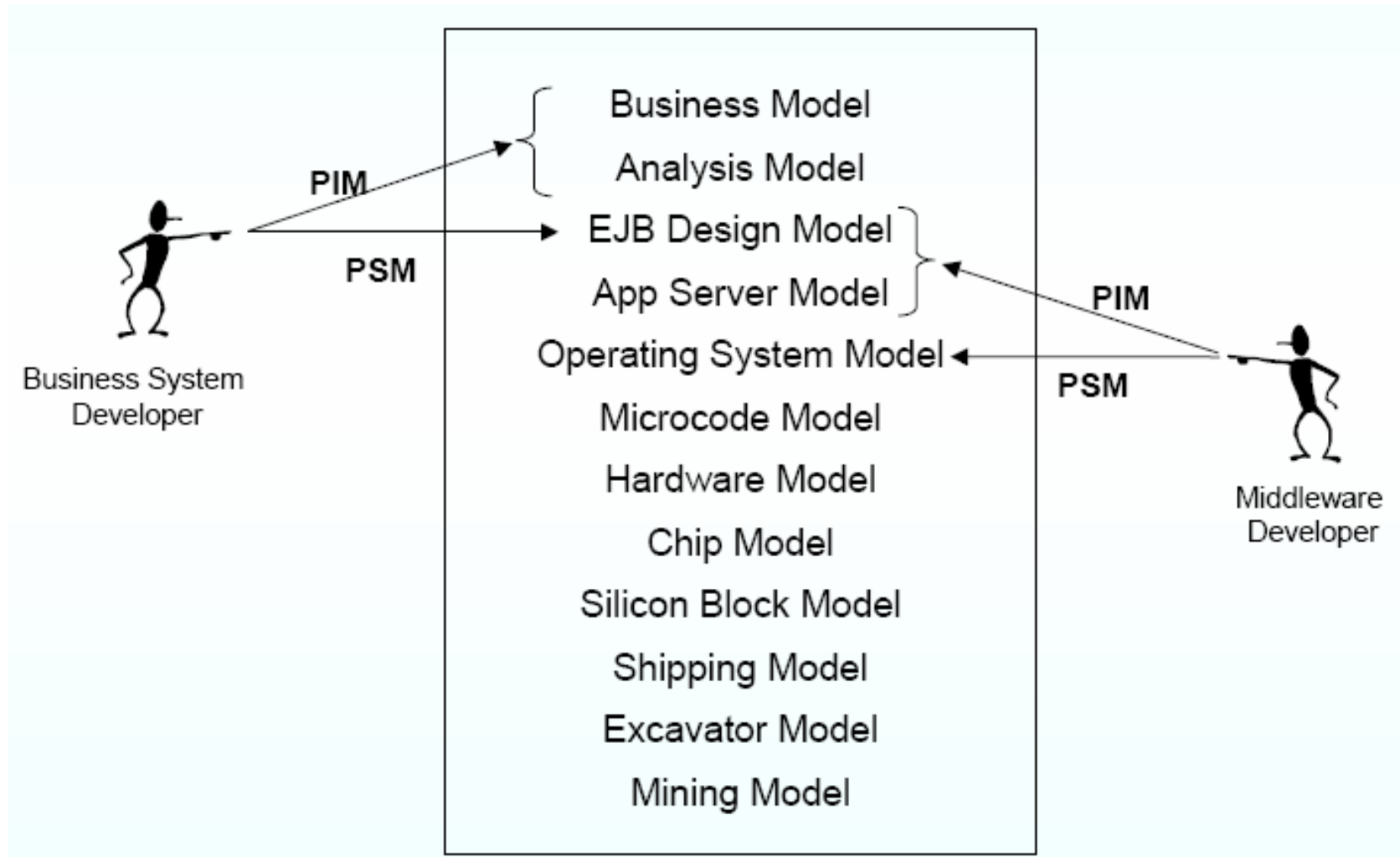
There can be more than one PSM, depending on the target system.

For SQL in ADL:

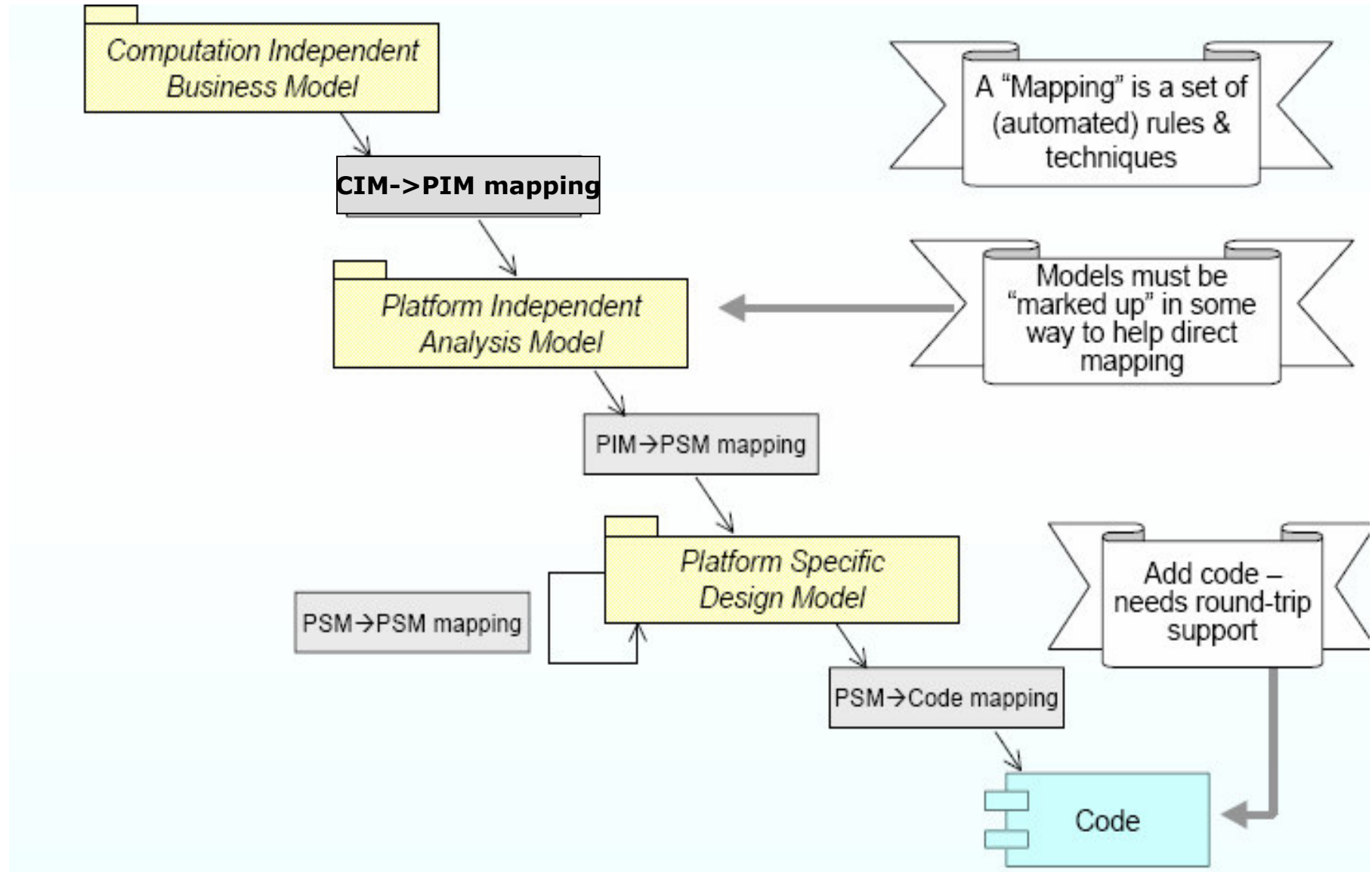
```
psm[SQL] University
from SQL import TABLE, VARCHAR
class Professor {
    concept
        characteristic name      :VARCHAR(100)
        relationship  lectures :TABLE(...)
        constraint uniqueNames PRIMARY KEY name
}
class Lecture {
    concept
        characteristic title     :VARCHAR(200)
        relationship  teacher  :VARCHAR(100)
        constraint fk_teacher FOREIGN KEY Professor(name)
}
```

# Importance of Context

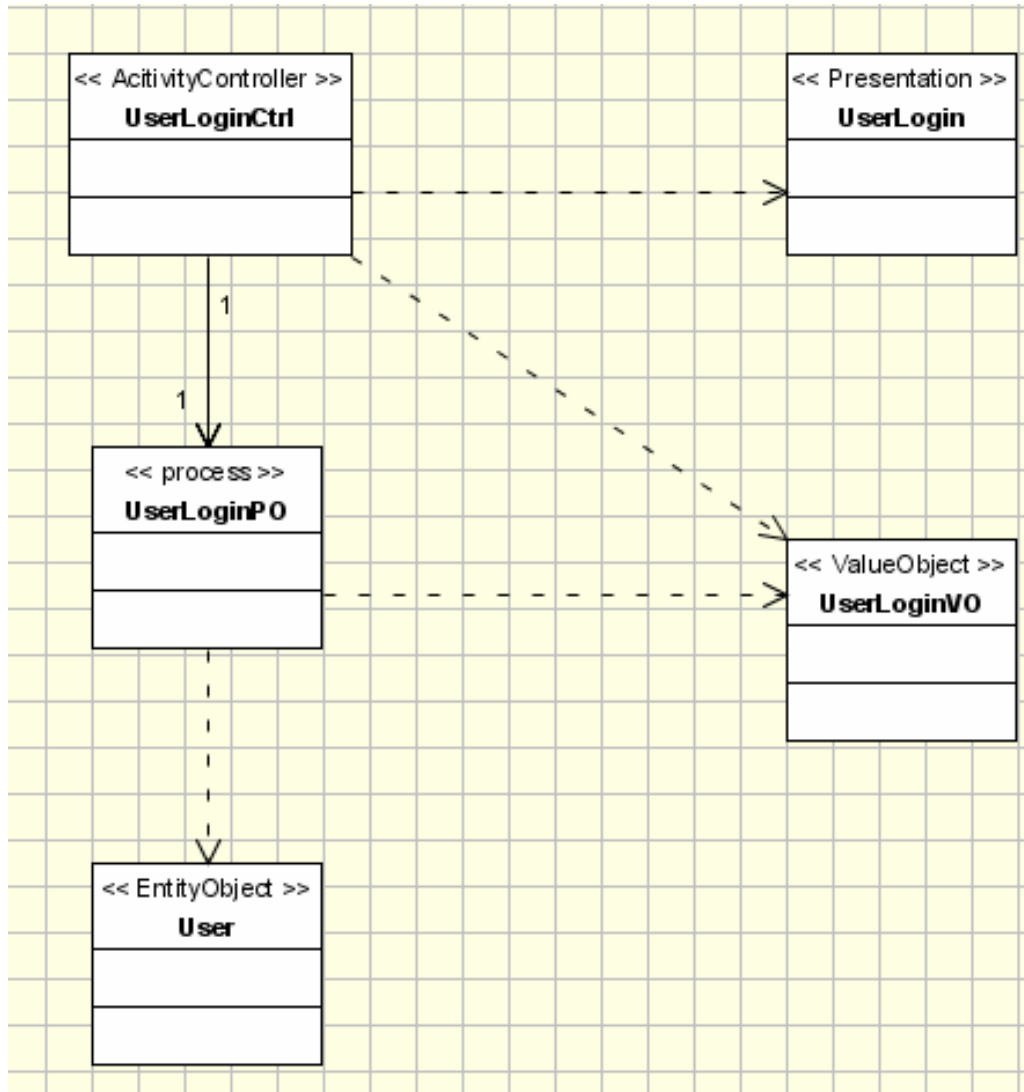
---



# Mapping

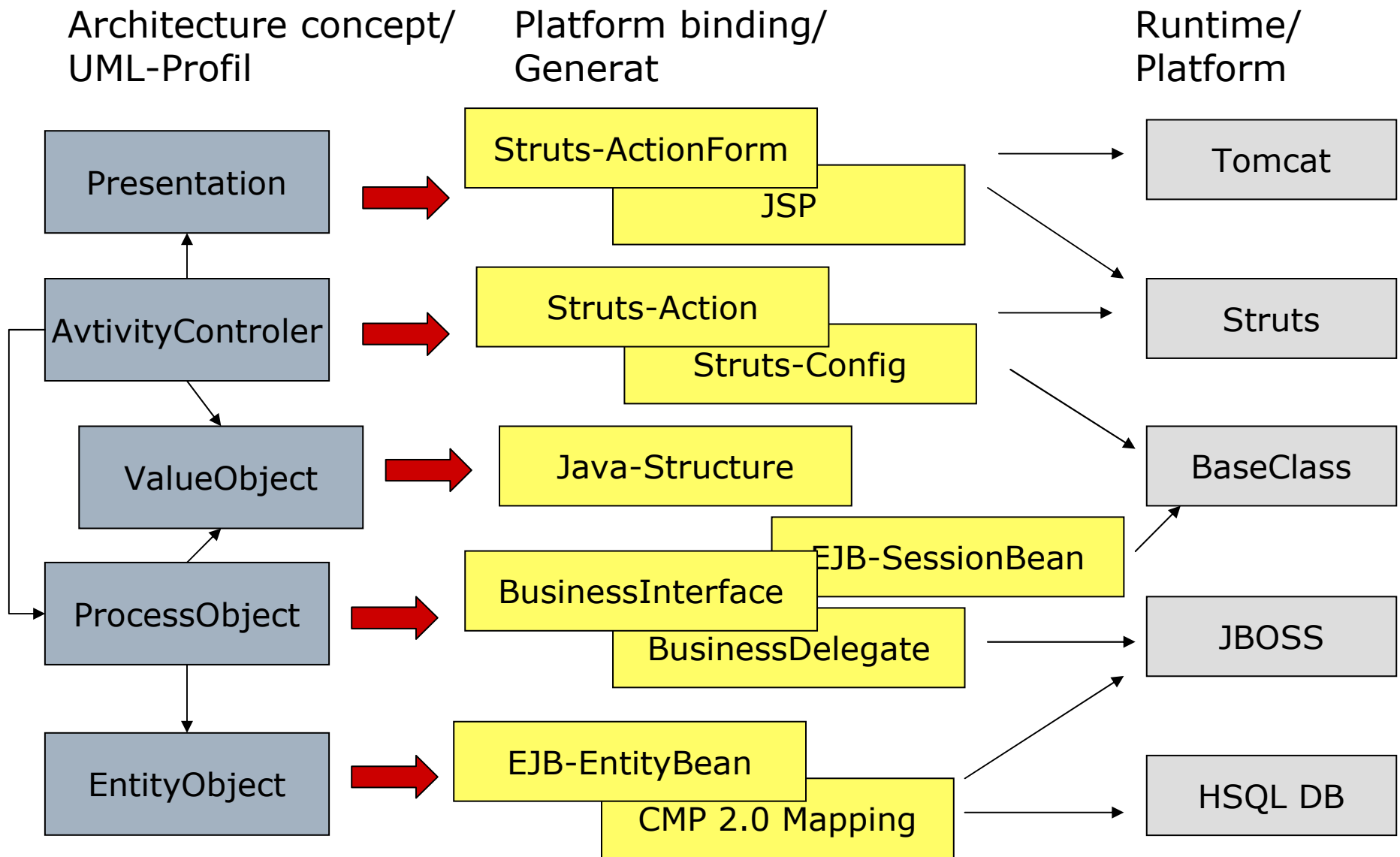


# AndroMDA view



AndroMDA is an extensible generator framework. Takes UML from tools as input (XMI). Generates deployable components based on mapping definitions. You can also write your own mappings (cartridges) to support your own architecture or framework.

# Platform/Runtime bindings



# Diagram Generation from Code

---

Code visualization, e.g.

- ❑ class diagrams from data models
- ❑ sequence diagrams from code

Needed for reengineering

- ❑ missing design documents are recreated (“invented”)
- ❑ e.g., to work with foreign code

Sometimes also used to implement roundtrip engineering

- ❑ if different tools are used for design and coding
- ❑ if one tool is used which is not capable of handling roundtrips
- ❑ if design documentation is unavailable.

# Diagram Generation from Code (2)

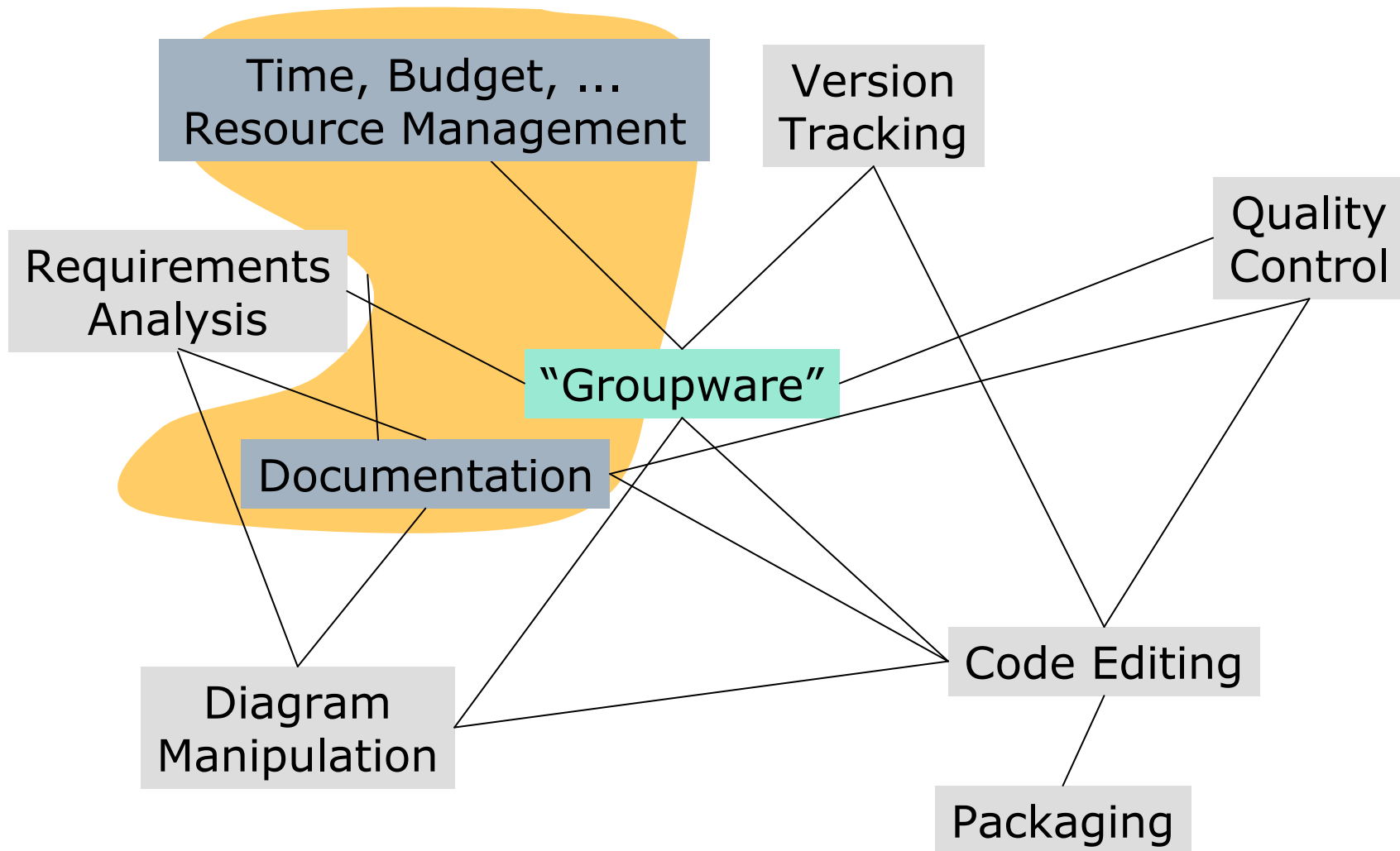
---

Common problems:

- Generated diagrams reflect physical code.  
Diagram generation does not recreate the exact same (conceptual) diagrams you created during design.
  - decisions made during transition from design to implementation are not reversible
  - patterns cannot be recognized in general
- Semantic mismatch between modeling and programming languages, e.g.,
  - no distinction between general associations and aggregations in Java
  - distinction of aggregation and composition is buried in code  
(remember: the distinction is a lifetime issue!)

# Reminder: Project Management Tools

---



# Project Management Tools

---

## A Software Manager's Responsibilities:

- Proposal writing
- Project budgeting
- Project planning and scheduling
- Project monitoring and reviewing
- Personnel selection and evaluation
- Report writing and presentation

## (Semi-) Formal Means:

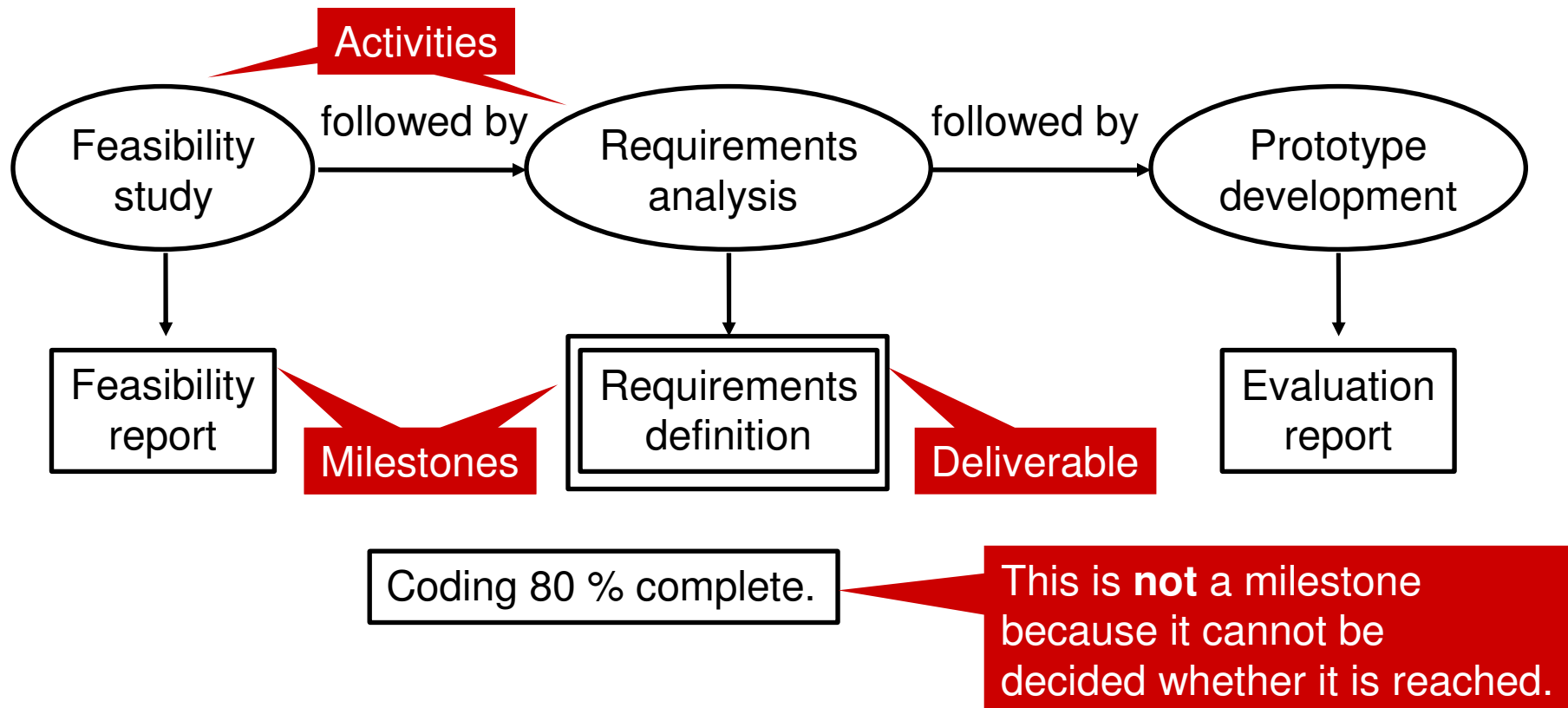
- Gantt diagrams (see MS Project)
- Network analysis (e.g., PERT chart)

# Activity Organization

---

**Milestone:** End-point of activity.

**Deliverable:** milestone delivered to customer.



# Task Duration and Dependencies

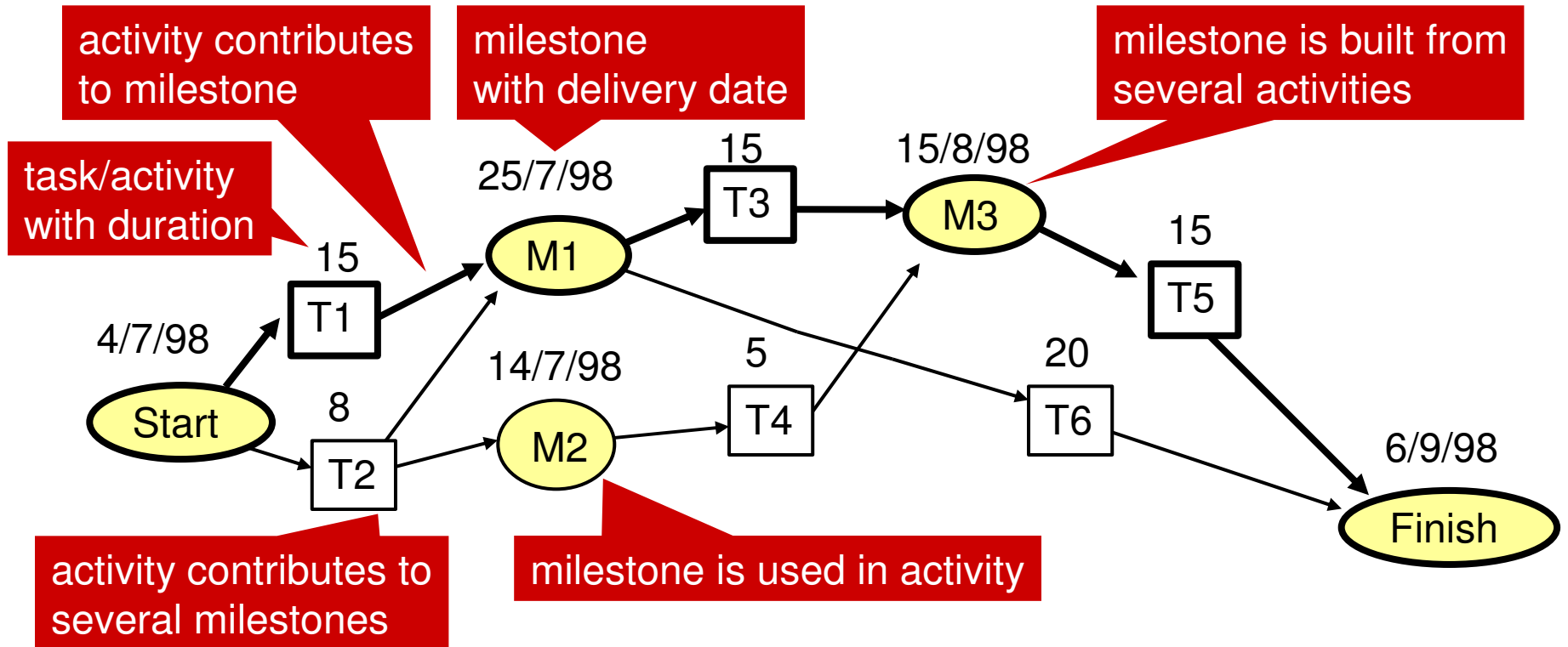
---

<b>Task/Activity</b>	<b>Duration (days)</b>	<b>depends on</b>
T1	15	
T2	8	
T3	15	T1, T2
T4	5	T2
T5	15	T3, T4
T6	20	T1, T2

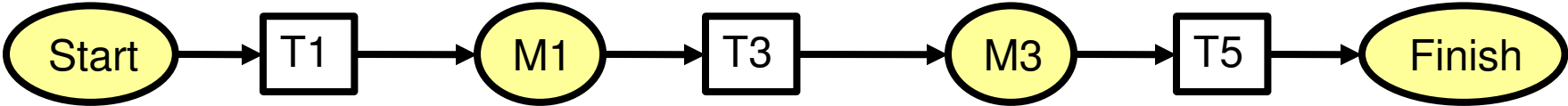
Typical duration of tasks/activities: 1-10 weeks

- finer grained (days): increases time needed for reviewing and re-scheduling
- coarser grained (months): decreases control and delay detection

# Activity Network / PERT Chart



**critical path:** any delay here causes delay of the project (no time buffer)



# PERT Chart

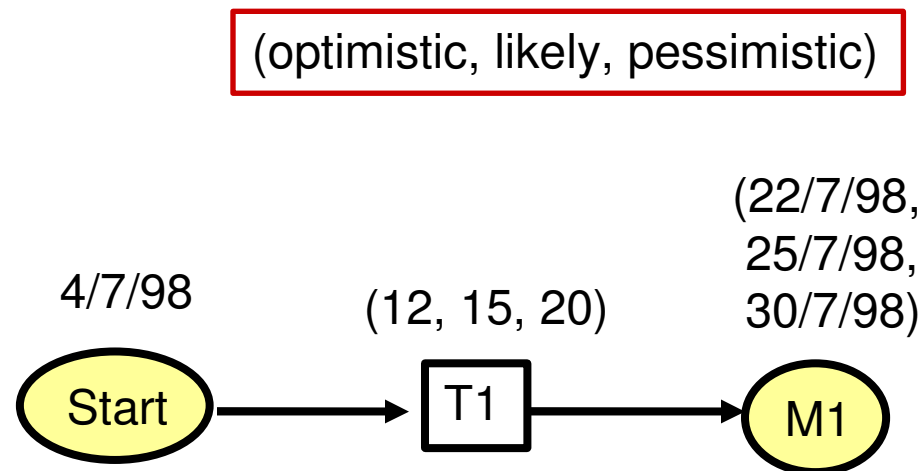
---

Sophisticated variant of activity networks

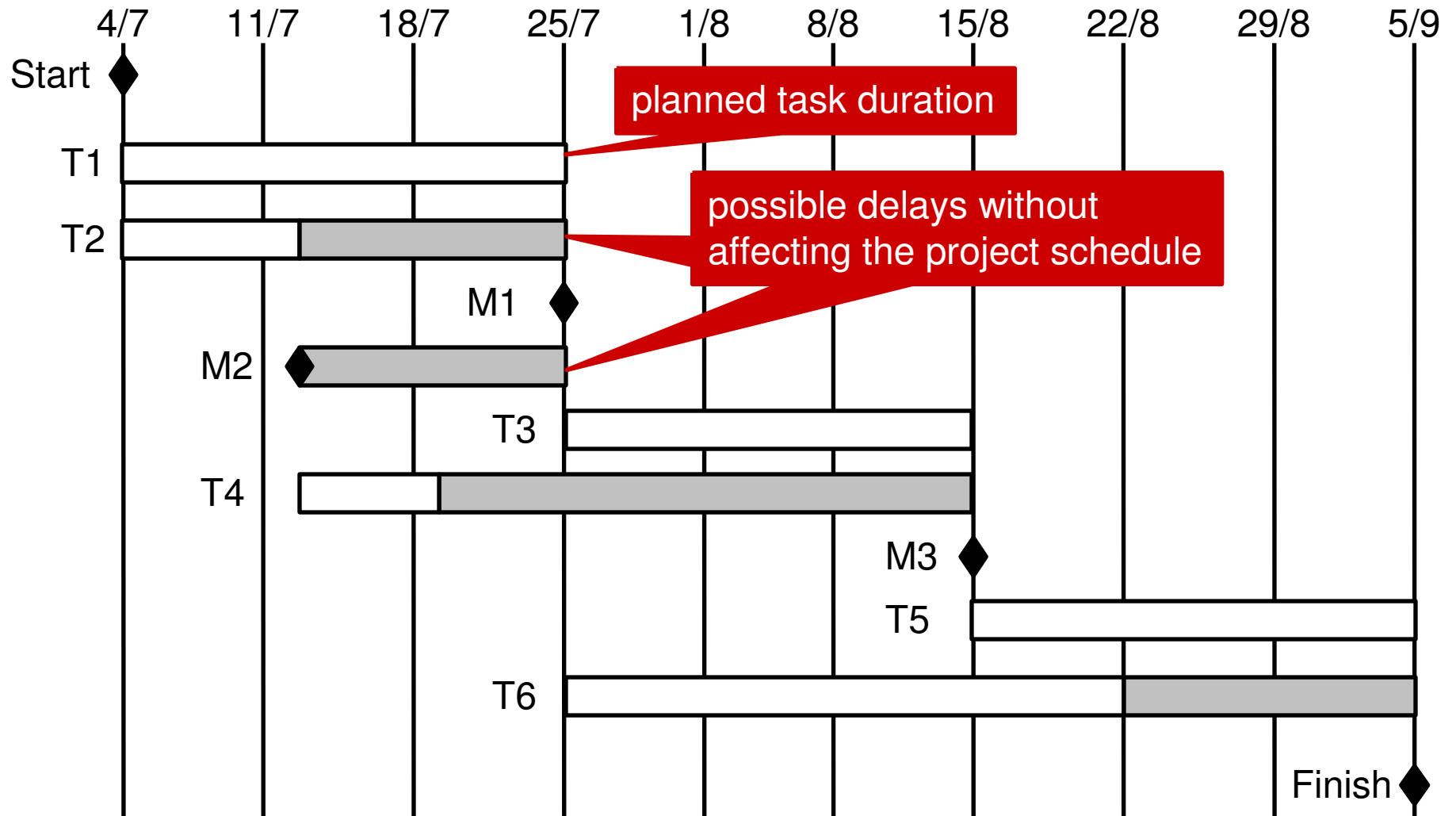
Distinguishes pessimistic, likely, and optimistic durations or dates

Leading to many potential critical paths

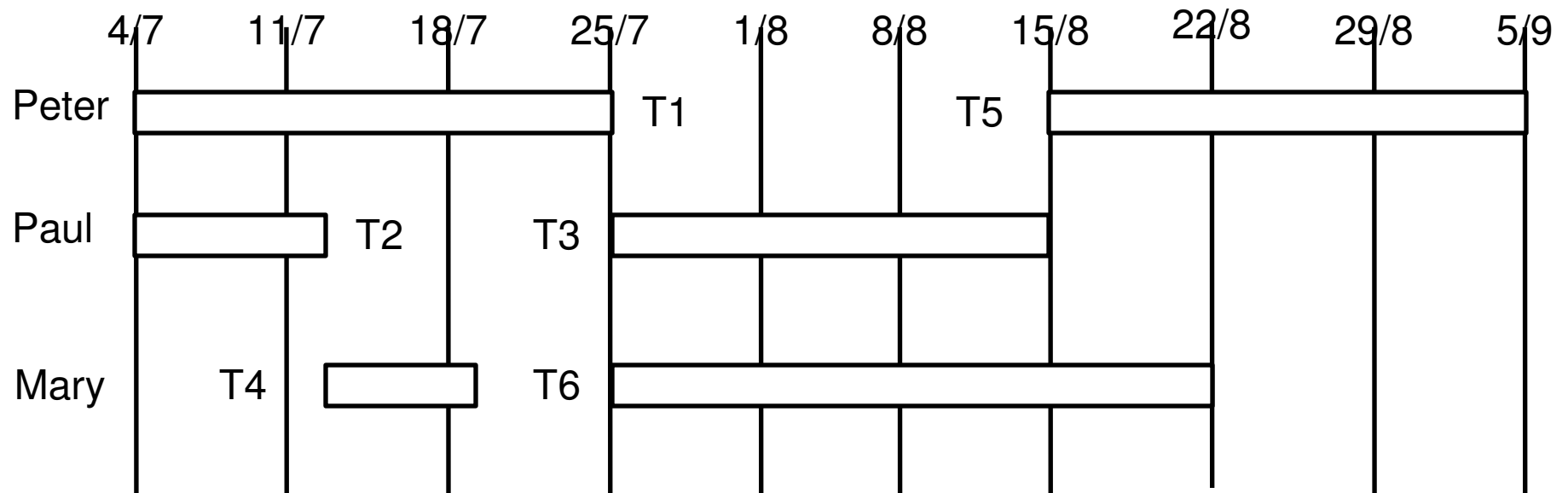
Critical path analysis only with tool-support



# Activity Bar Chart / Gantt chart



# Staff Allocation



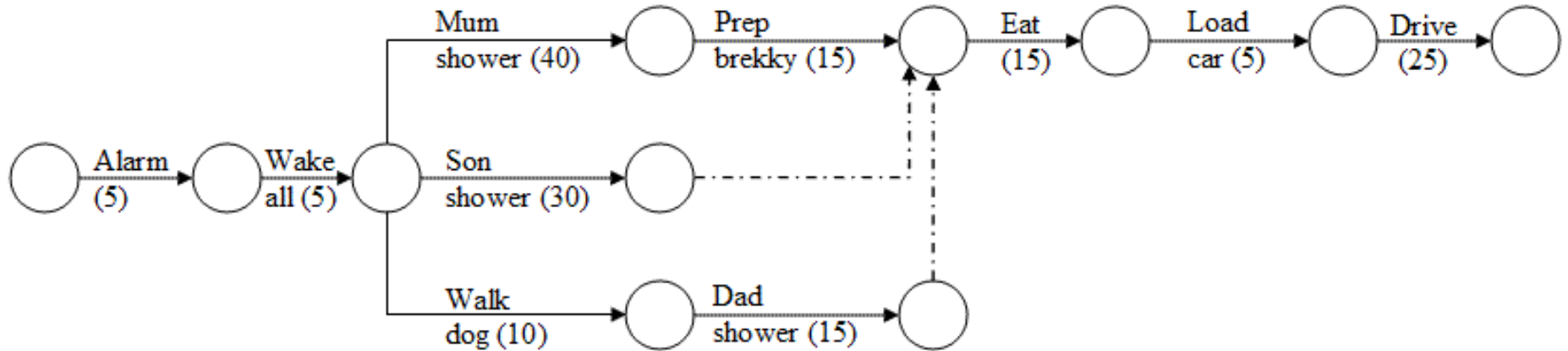
Include time for

- holiday,
- training courses, and
- other projects.

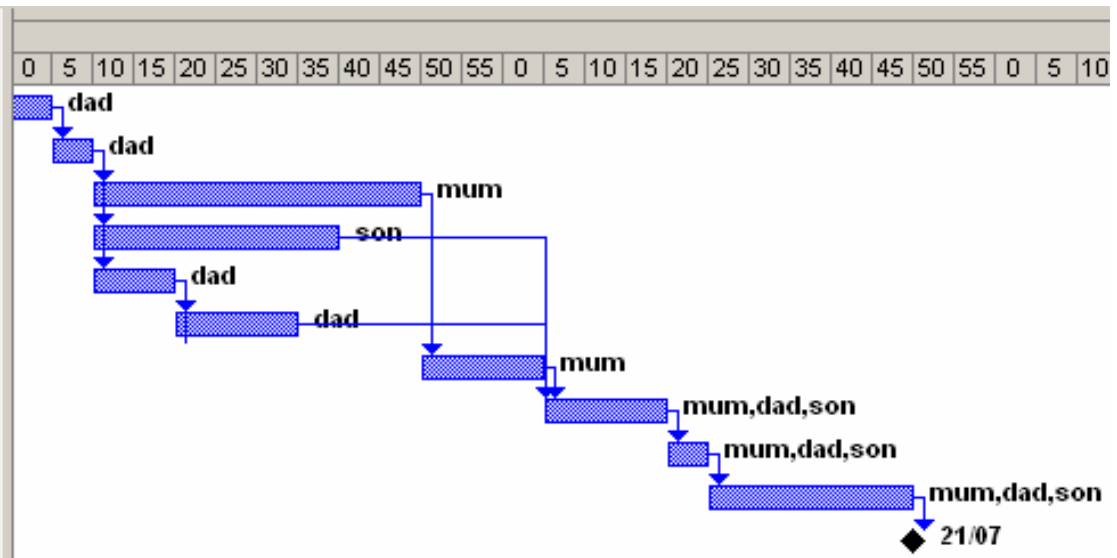
Be prepared to handle delays:

- people may be unavailable and
- specialists cannot be replaced on short notice.

# A Family Routine

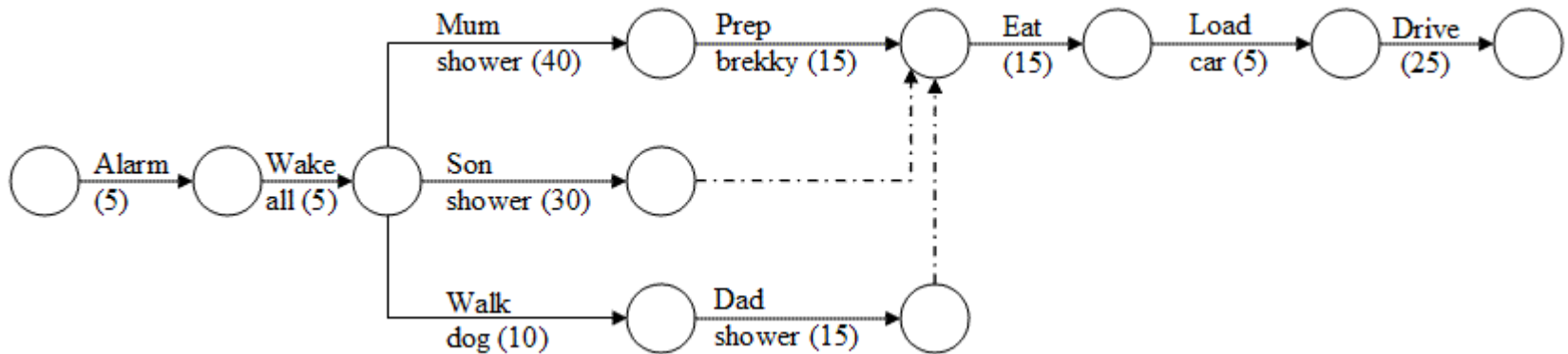


	Task Name	Duration	Precede	Resource Names
1	alarm goes off	5 mins		dad
2	wake family	5 mins	1	dad
3	mum shower	40 mins	2	mum
4	son shower	30 mins	2	son
5	walk dog	10 mins	2	dad
6	father shower	15 mins	5	dad
7	prepare breakfast	15 mins	3	mum
8	eat breakfast	15 mins	8,4,6	mum,dad,son
9	load car	5 mins	9	mum,dad,son
10	drive to game	25 mins	10	mum,dad,son
11	arrive at game	0 mins	11	



# Questions

---



**Q1. What tasks are on the critical path?**

**Q2. What is the *minimum* time it would take for the family to reach the footy game after getting the alarm goes off?**

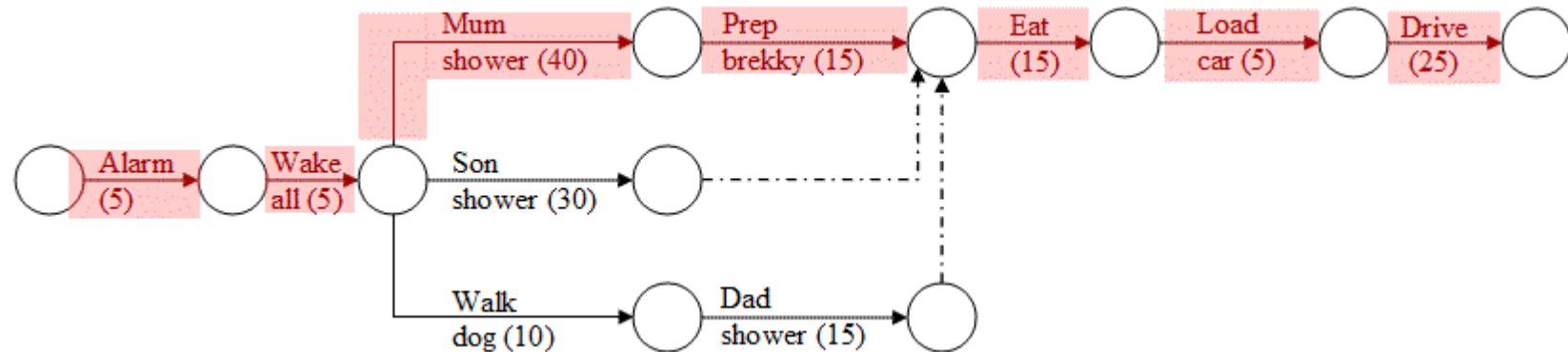
**Q3. How much more time could dad walk the dog before eating breakfast got delayed? (Note: Mum insists the entire family eats together)**

**Q4. What is this amount of time called?**

**Q5. If mum skipped her 40 minutes shower, how much earlier would they get to the game?**

# Q1

---



What is the critical path?

**Path 1 = 5+5+40+15+15+5+25 = 110 min**

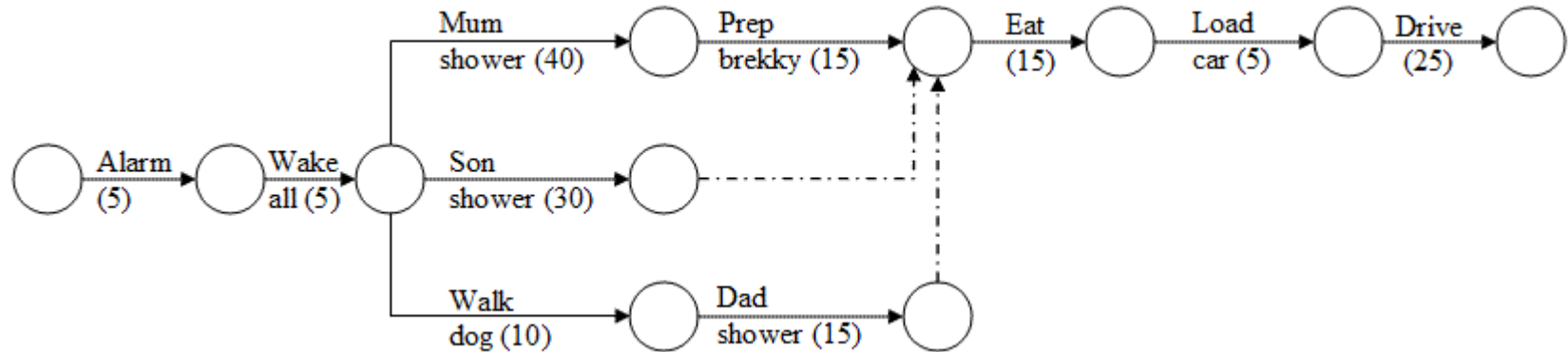
**Path 2 = 5+5+30+15+5+25 = 85**

**Path 3 = 5+5+10+15+15+5+25 = 80**

**The critical path is the longest path : path 1**

# Q2

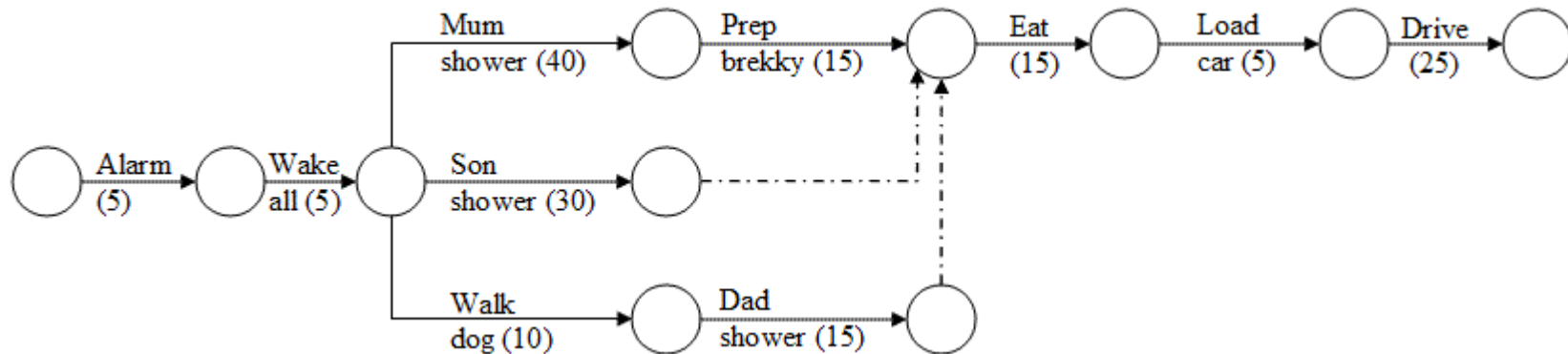
---



What is the minimum time it would take for the family to reach the footy game after the alarm starts ringing?

**The duration of the critical tasks... 110 minutes**

# Q3 & 4



How much more time could dad walk the dog before eating breakfast got delayed?

**30 minutes...**

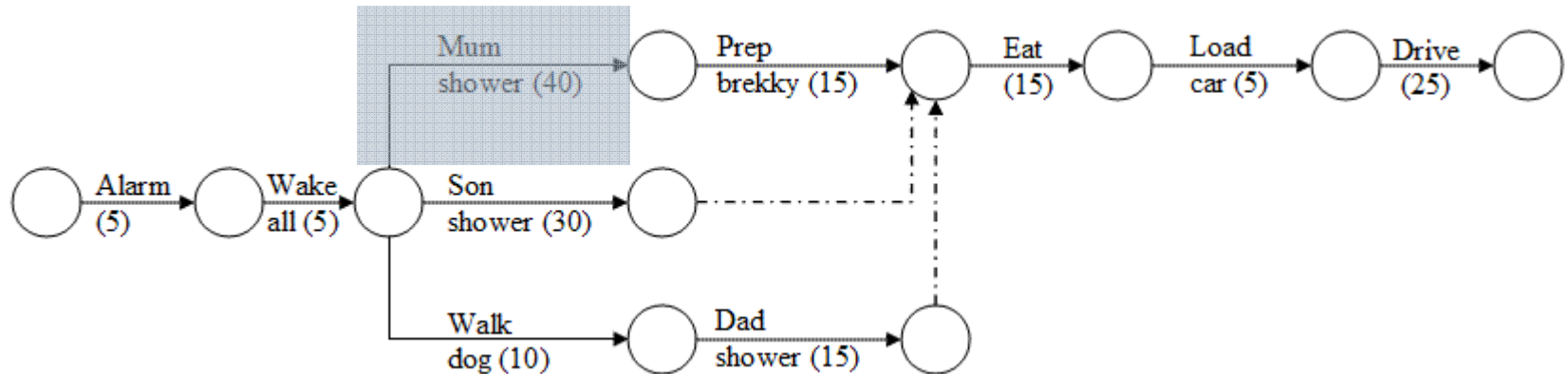
***Shower + Prep Brekky = 55 min vs***

***Walk Dog + Dad Shower = 25 min ... 30 min diff***

What is this amount of time called?

**Slack time (or float)**

## Q5



If mum skipped her 40 minute shower, how much earlier would they get to the game?

**When the critical path is reduced by 40 minutes, it stops being the critical path. Path 2, at 85 min, becomes the critical path. Since it is 25 min shorter than the original 110 minute critical path, there is a 25 minute saving.**

# References

---

## Software - UML Tools:

- Together
- Rational Rose
- Gentleware Poseidon / ArgoUML
- Omundo
- Microsoft Visio
- Magicdraw UML

## Books

- Three Amigos; three books
- Fowler: UML distilled

## Articles

# References (2)

---

## Software - MDA Tools:

- ArchStyler
- IBM XDE
- OptimalJ
- AndroMDA

## Articles MDA:

- Introduction to MDA  
<http://www-106.ibm.com/developerworks/rational/library/3100.html>
- OMG's MDA site  
<http://www.omg.org/mda/>