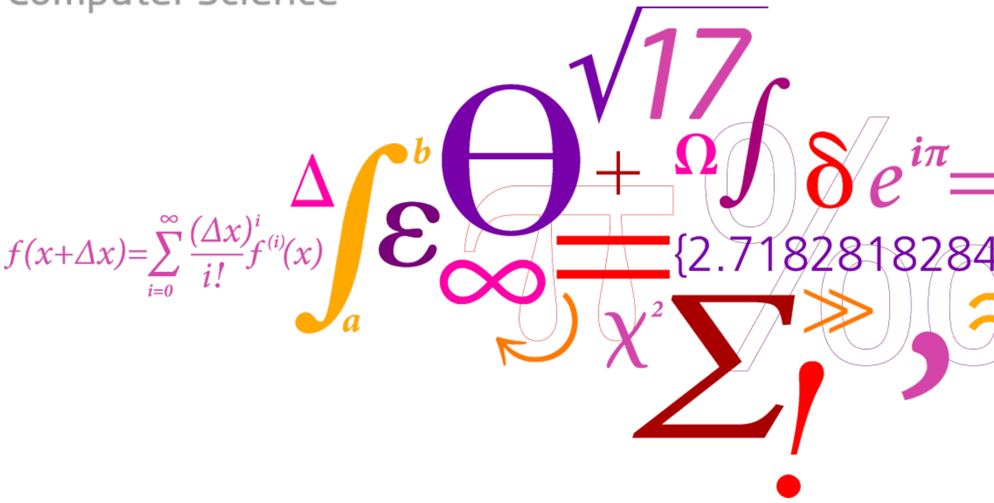
Software Engineering 2
A practical course in software engineering

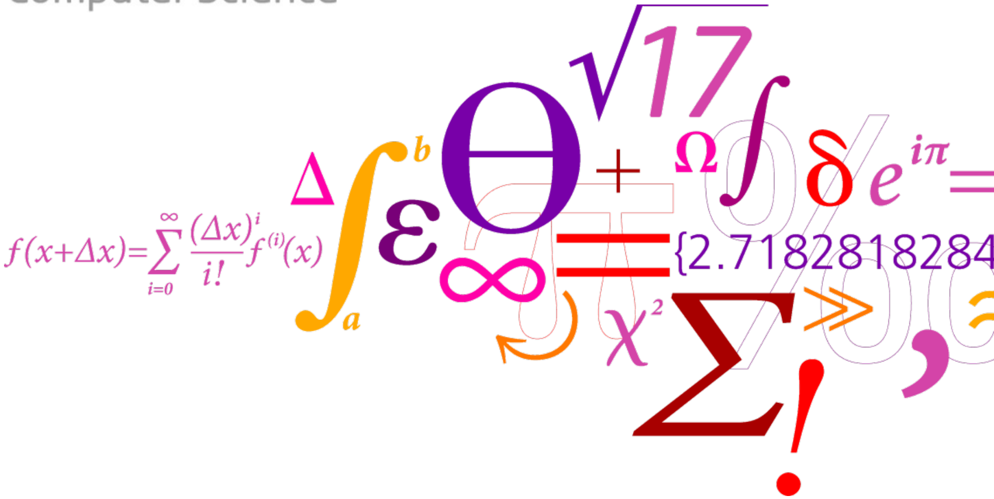
Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science


$f(x+\Delta x)=\sum_{i=0}^{\infty}\frac{(\Delta x)^i}{i!}f^{(i)}(x)$
$\sqrt{17}$
$\{2.7182818284$

I. Introduction

DTU Compute
Department of Applied Mathematics and Computer Science



Introduction

- Motivation: Software engineering & management
- Agile development
- The role of models in software engineering

- Organisation of this course

- Project (and tutorials)
  - The task
  - Organisation
  - Forming the groups

Weekly Schedule (roughly)

	Mon	Tue	Wed	Thu	Fri
8-10		lecture			
10-12		🟦🟧🟪			
13-15					🟦🟧🟪
15-17					project

Plus actual work on the project!!

🟦 lecture

🟧 tutorial

🟪 project

There will be many exceptions from the rule! See web pages for details!

1. Motivation

- Objectives of this course:
  Skills in software engineering!

- What is “software engineering”?
- What is “software”?

- software ~~=~~ program
- software engineering ~~=~~ programming

Program vs. Software

is much more than

Software >> Program

Software Engineering >>> Programming

is much much more than

Programmi

Software

Software Engineer

Software Engineering

If somebody has built a garage, would we let him built a skyscraper? No, never!

If somebody has written a program, would we let him built software? Yes we would!

Program

Programmer

Programming

But, of course we should not!

Software Engineering is

… much more than programming!

… listening and understanding!
… analytic and conceptual work!
… communication!
… a social process!
… acquiring and using new technologies!
…

but, still evolving

… a discipline with proven concepts, methods, notations, and tools!
… and ever new technologies emerging!
and needs discipline

Experience

Software Engineering requires much experience!

This experience

- can not be taught theoretically!
- will be provided in this course!

→ project

→ tutorial (new technologies)

→ and (only) backed by the lectures

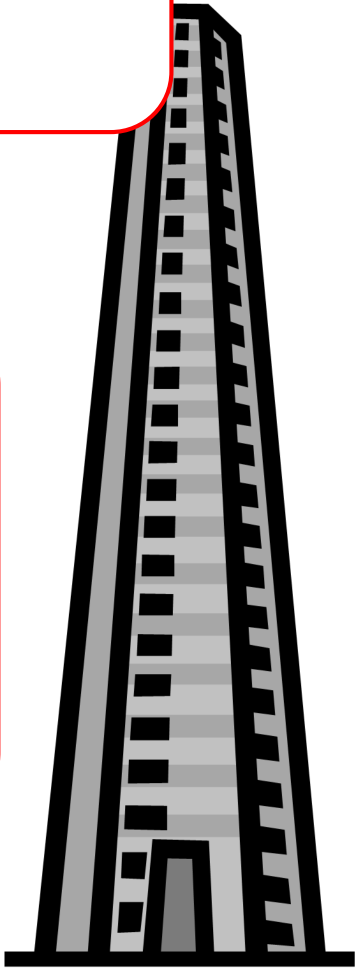
Due to the special nature of this year's project, with a special focus on agile development!

Analogy revisited

Effort per participant

- 10 ECTS = ca. 270h work
- ca. 20h/week

The experience of a big project cannot be replaced by the experience of many small ones.



Objective

Practice the concepts, methods, notations and tools for software engineering

- improve programming skills
- understanding of the software engineering process
- agile practices
- experiences with problems and concepts for solving them
- writing and creating documents and models
- use of methods and tools
- practice communication and presentation skills
- capability of teamwork and leadership
- acquire new technologies
- …

If in doubt:

1. think
2. search (“google”)
3. ask

Excursion: CDIO

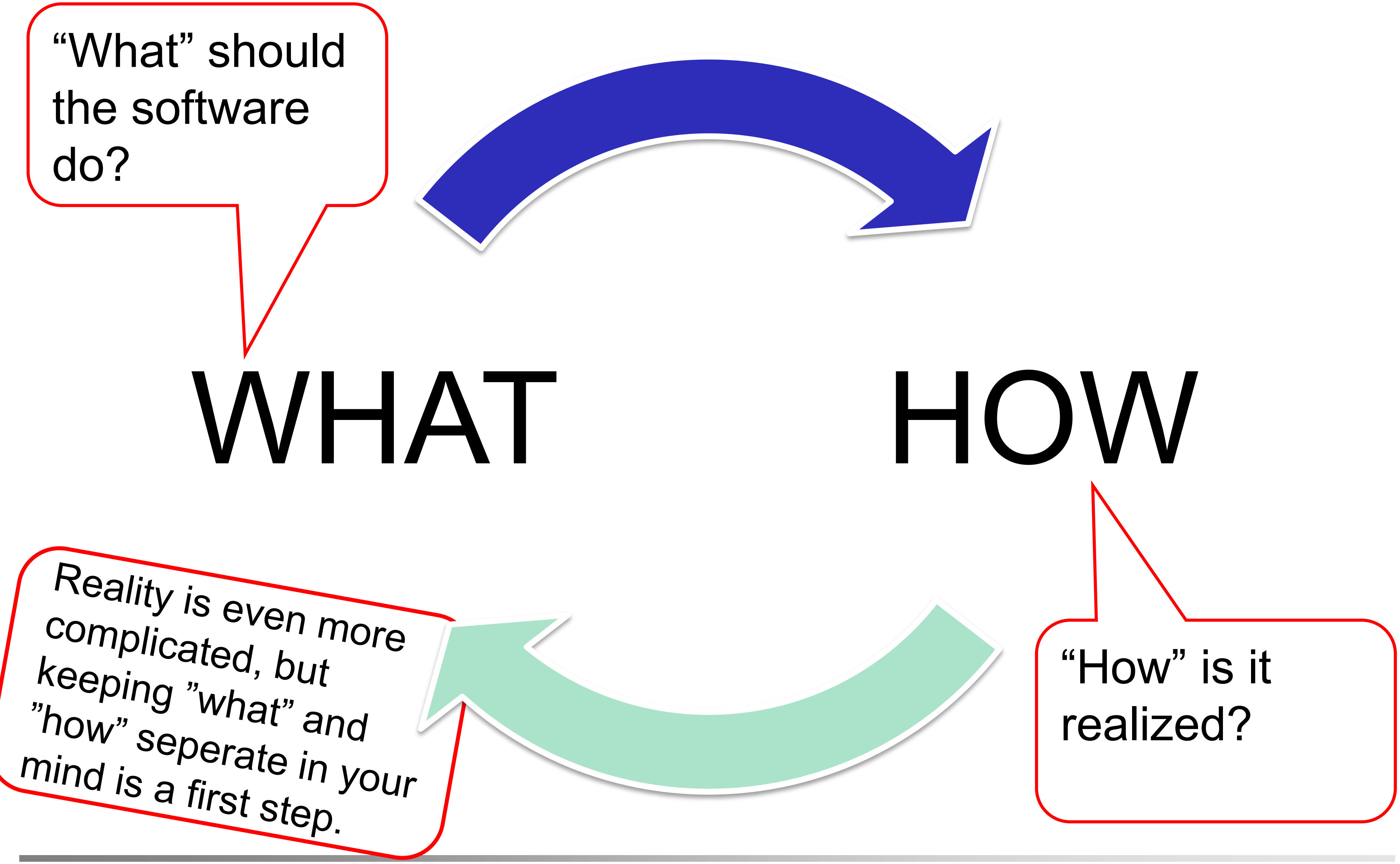
Conceive

Design

Implement

Operate

Co-evolution



“What” should the software do?

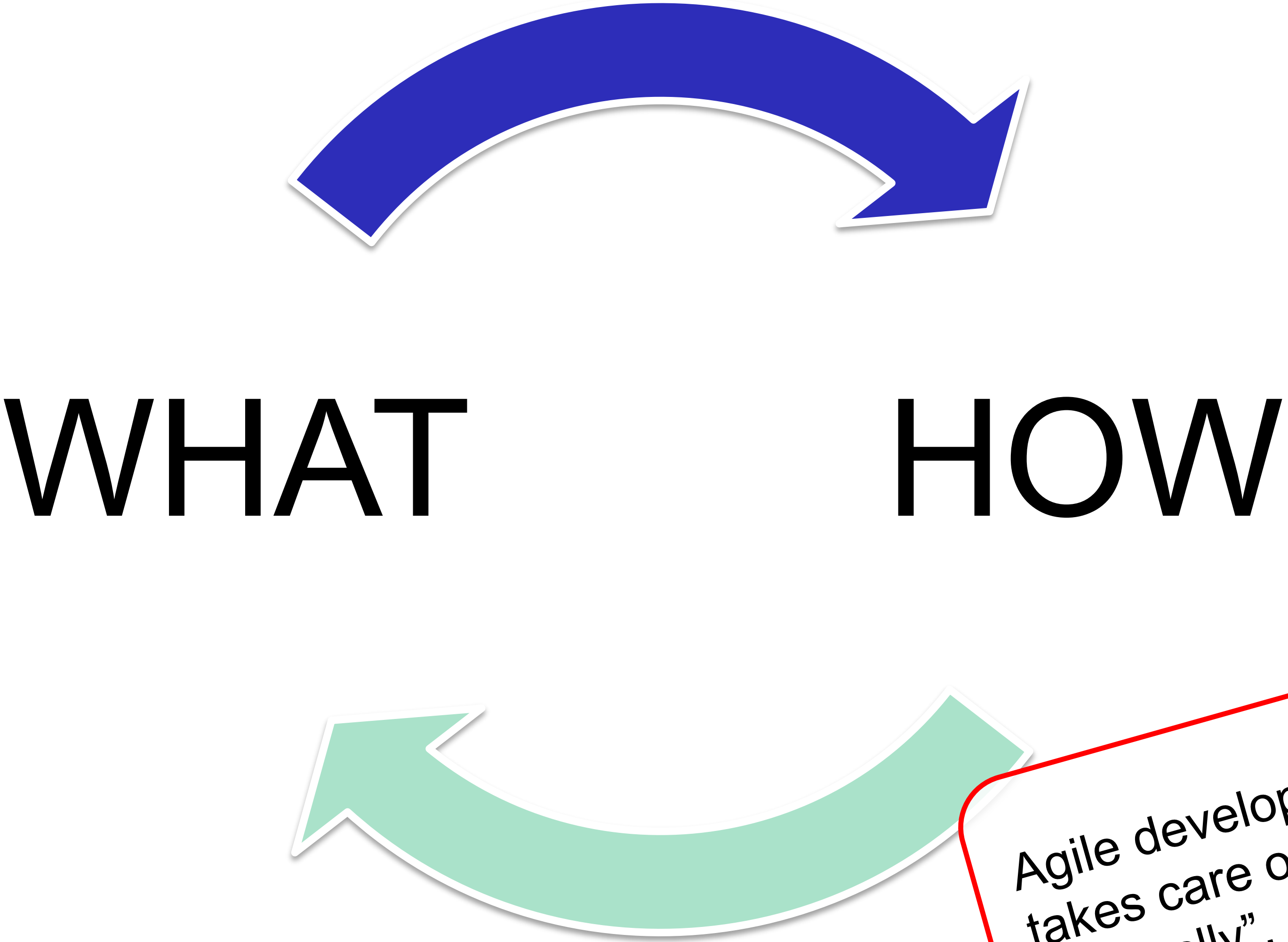
WHAT

HOW

“How” is it realized?

Reality is even more complicated, but keeping ”what” and ”how” seperate in your mind is a first step.

Co-evolution



WHAT

HOW

Agile development takes care of this cycle ”naturally”.

Questions

- Why do so many software projects fail?
- Why is software development so hard (or at least harder as we believe)?
- BTW: What is software?

Software

The sum of all **programs**, **procedures** and **objects** along with the associated **data** and **documentation**, which are necessary (or at least desirable) for running an application on a computer system.

[free translation of the German Informatik DUDEN and Hesse‘s definition]

Software …

is becoming more and more complex!

Exponential growth of software (in „lines of code“ LOC) within the same product line:

- Apollo (NASA‘s Apollo programme)
- Cars (automotive software)
- …

Technology needs to keep up with that!

Software …

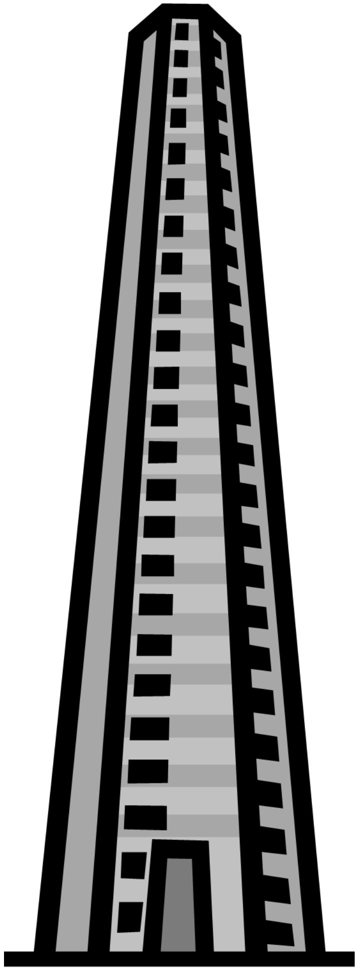
cannot be „programmed“ by a single person anymore; a single person cannot fully comprehend all the details of software any more.

Efforts of 10 to 100 person years (PYs) are quite standard in software development.

Software …

is intangible.

You cannot touch, see or feel software. Humans lack a “natural feeling” of software and its complexity.



Software …

does not wear out,
but becomes of age anyway
(in relation to the environment it is running in and the expectations of the end user)!

Software needs „maintenance“! But, this does not mean the same as in traditional engineering (e.g. in mechanical engineering, where systems physically wear out).

Actually, maintenance is a big factor in the cost of IT systems.

Software …

„lives“ longer than its creators expected it to live.

Y2K

Software …

is everywhere and many lifes depend on it.

Software Engineering is

… much more than programming!

… listening and understanding!

… analytic and conceptual work!

… communication!

… a social process!

… acquiring new technologies!

Problems

- imprecise requirements
- mistakable and unclear requirements
- inconsistent requirements
- changing requirements

- changing environments (software / hardware)
- different versions and configurations
- changing tools, notations, languages, methods, concepts, technologies

- collective knowledge only
- communication
- …

Software engineering

is the sum of all means, facilities, procedures, processes, notations, methods, concepts for developing, operating and maintaining a software system.

Software engineering

Branches:

- **Development**:
  actual development of the software product
- **Management**:
  Manage (control and improve) the development process
- **Quality management**:
  Planning and implementing measures that guarantee that the software meets the required quality
- **Software maintenance**:
  Remove faults occurring in operation, adapt software to changing requirements and environments

Process models

Process models (life cycle models) are the „distilled“ experience of successful software projects.

They define a functional procedure along with appropriate documents.

▪ **What** should be done	document, notation
▪ **when**,	phase
▪ by **whom** and	role
▪ **how**!	method

Process models

Problem: Often process models are used very mechanical and in a „meaningless“ way.

- → documents just for the sake of the process
- → (UML) diagrams just for the sake of UML
- → comments just for the sake of comments

Therefore:
1. Think!
2. What is reasonable?

Rule of thumb

When producing and compiling a document, ask yourself:

- What should the document be good for?
- Who should be addressed?
- Which information is expected?
- What is the common „pragmatics“?
- …

Here, document can also be code including comments.

In short: What is reasonable?

Lectures and discussions will give some guidelines, though!

2. Agile Development

- Since we use agile development from day 1, we discuss the motivation of agile development and the used practices already today.

Only briefly today.

- We will provide some details, the theoretical underpinning and justification later (next week and again towards the end of this course)

Agile manifesto

„We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Very effective, but requires discipline!

That is, while there is value in the items on the right, we value the items on the left more.“

Manifesto for Agile Software Development,Kent Beck et al. 2001

Agile Practices

- We will talk about the *values*, *principles*, and core *concepts* and *activities* of agile development later in this course (week 2)
- For now, we discuss the core agile *practices* used in this course

Agile Practices

- **On-site customer** (Ekkart and sometimes staff and partners from the LiRA project)
- **Small/short releases** 2-3 week (see schedule)
- **Planning game** (based on User Stories for next release)

More details on Friday (informal planning game)!

Agile Practices

- **Coding standards**
- **Testing** (automated unit test)
- **Continuous integration** (use of Git and Jenkins)

"Release 0" in week 4!

Agile Practices

- **Pair programming** (all code developed and checked in by two persons)
- **Simple design**
- **Refactoring**

Status reports (SR)

In some tutorial/project sessions on Friday (13-15)

status report by each group (5-10 minutes each)

- can include brief code reviews and
- demos of running software (CI) and
- retrospective:
  - what went well
  - what did not go well
  - what can we do about it
  - what's up next

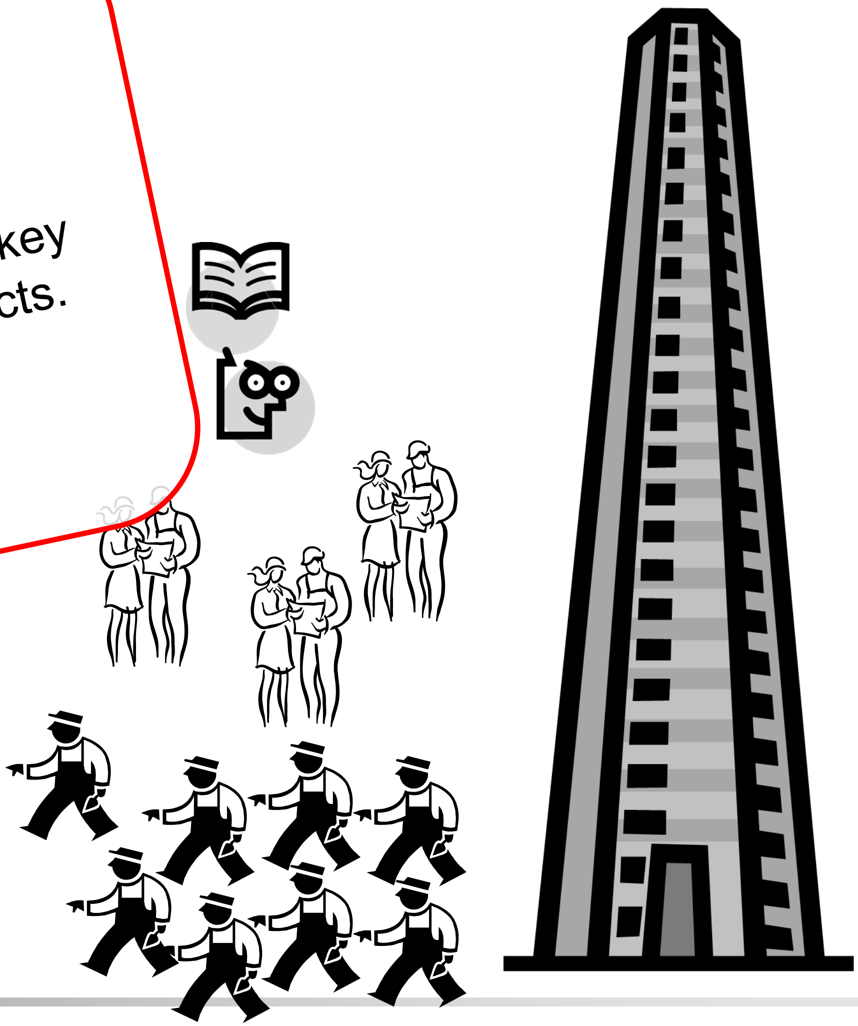
Starting in week 3

3. Models in SE

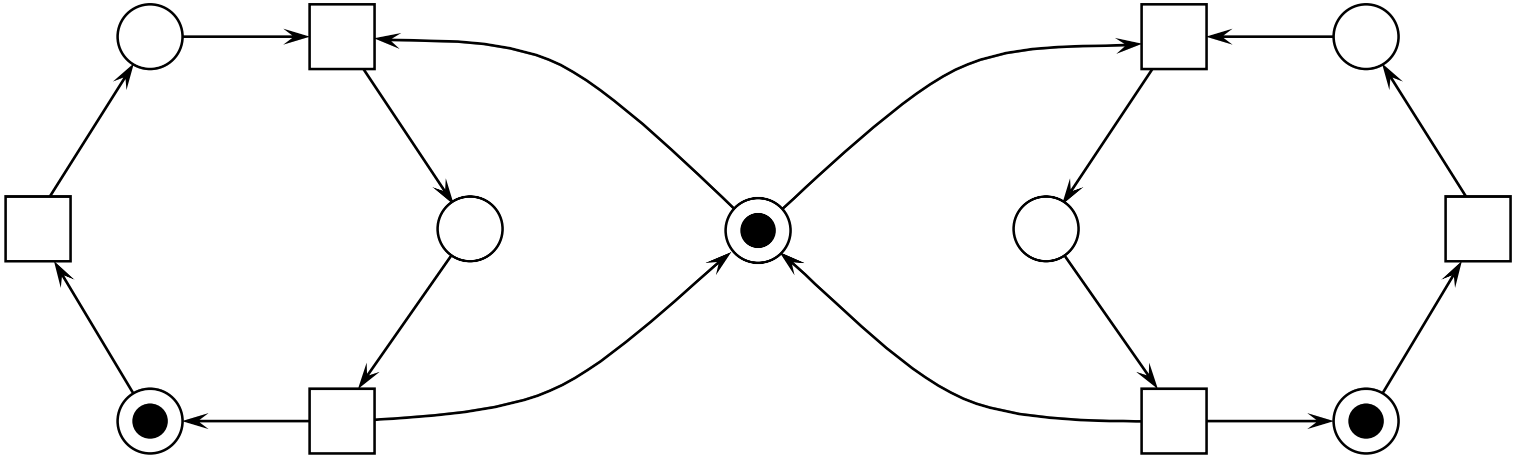
… and a glimpse of how software can be developed by using models – without doing any programming at all.

Modelling

Models are the “floor plans” of software engineers, and are the key to the success of software projects.



A Model (Petri net)



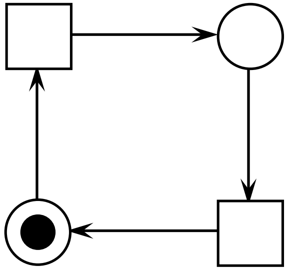
In the following, let us pretend we want to implement an editor for Petri nets and better understand what Petri nets are (the domain in this example project).

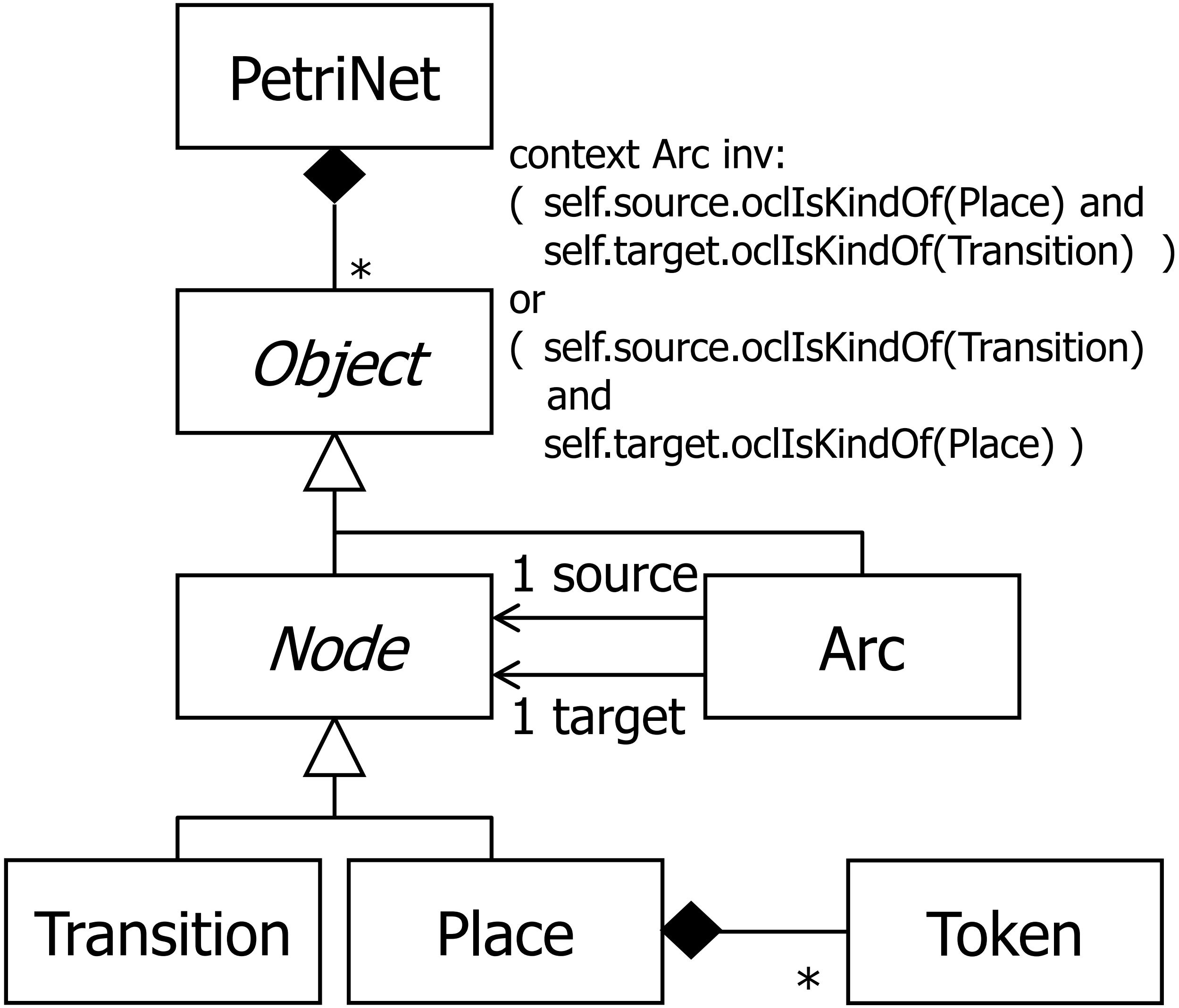
Stages

- Examples
- Taxonomy (done on blackboard)
- Glossary
- Model (see next slide)

Rule: Never ever start making a domain model without having seen some examples first and naming the main concepts (taxonomy)!

Models and Meta Models

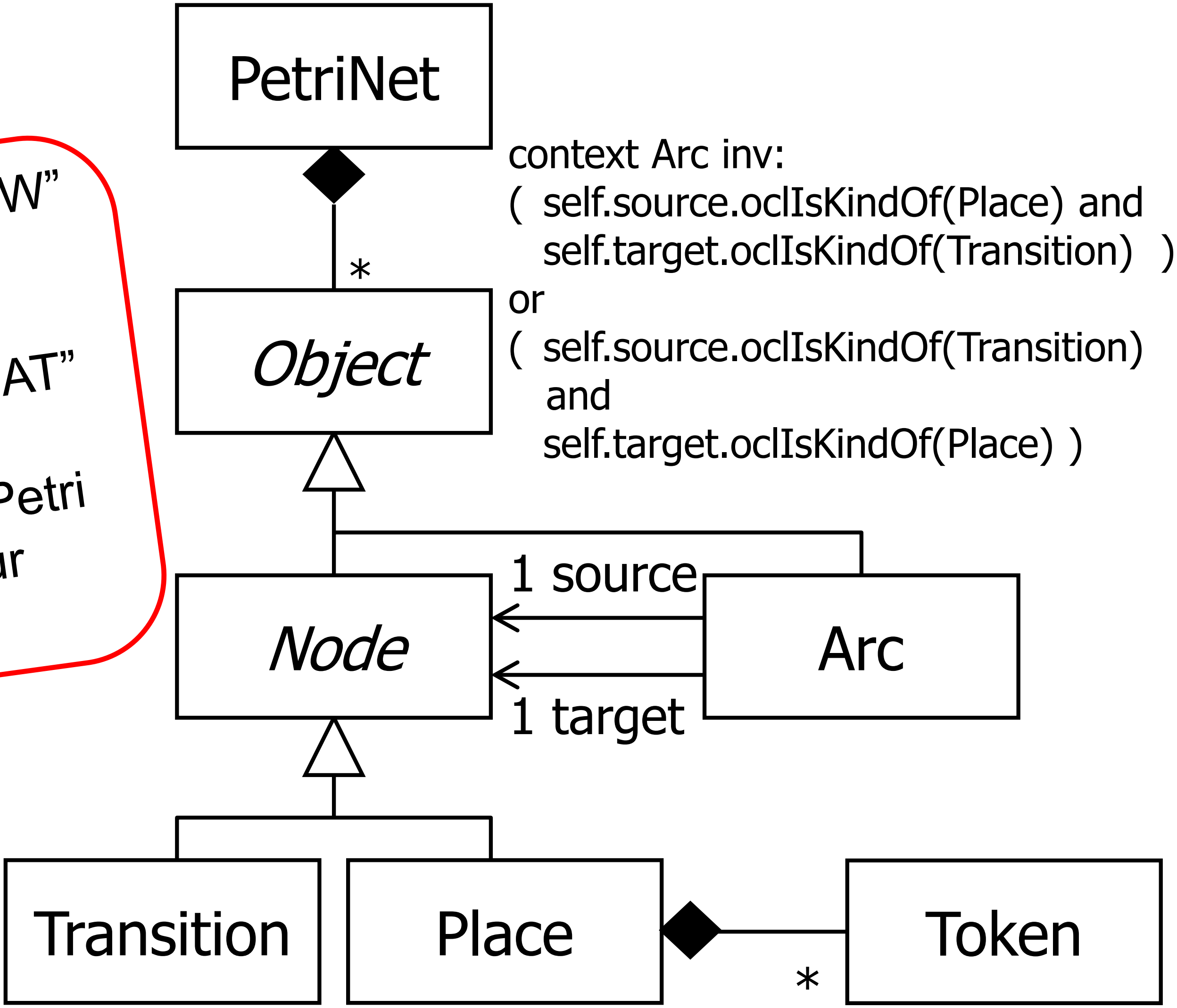




```
context Arc inv:
( self.source.oclIsKindOf(Place) and
  self.target.oclIsKindOf(Transition) )
or
( self.source.oclIsKindOf(Transition)
  and
  self.target.oclIsKindOf(Place) )
```

Meta model for Petri nets

Don’t think models as Java



context Arc inv:
(self.source.oclIsKindOf(Place) and
self.target.oclIsKindOf(Transition))
or
(self.source.oclIsKindOf(Transition)
and
self.target.oclIsKindOf(Place))

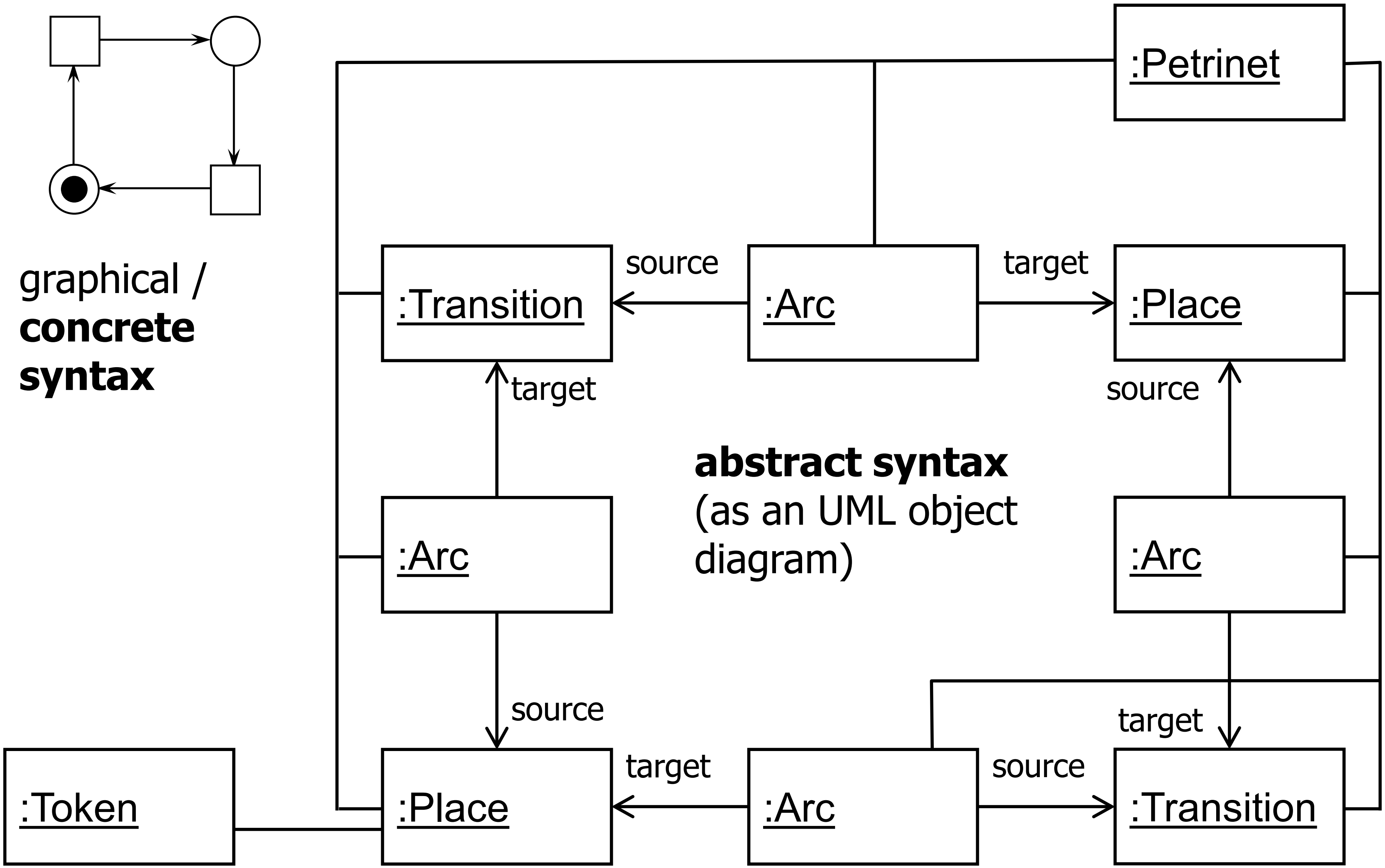
Rule: Don’t think of ”HOW” (programming) for now!

These models are ”WHAT” (domain) only: which concepts are there in Petri nets (the domain of our example)?

We will do that with examples of the domain of our project several times during this course!

Groups should have a working document for the domain of our project.

Syntax (abstract and concrete)



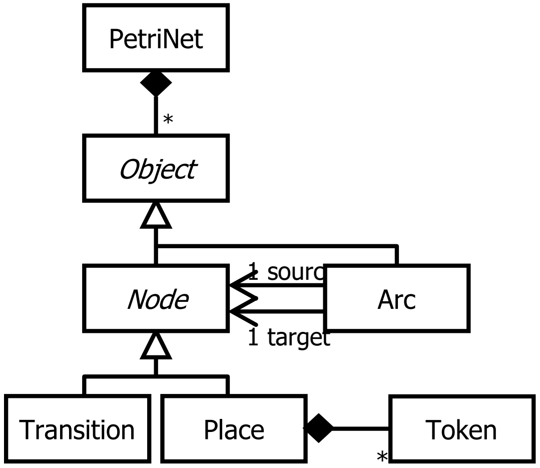
graphical / **concrete syntax**

abstract syntax (as an UML object diagram)

Overview

meta model

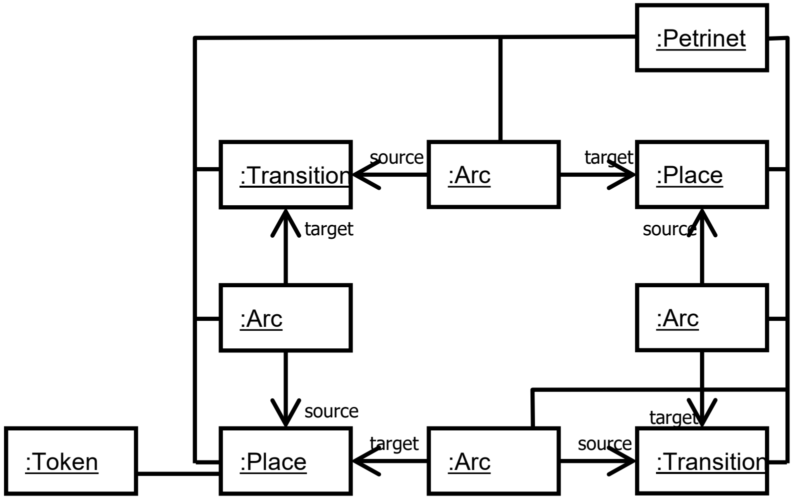
build-time



is an instance of

model

runtime



Benefits of Modelling

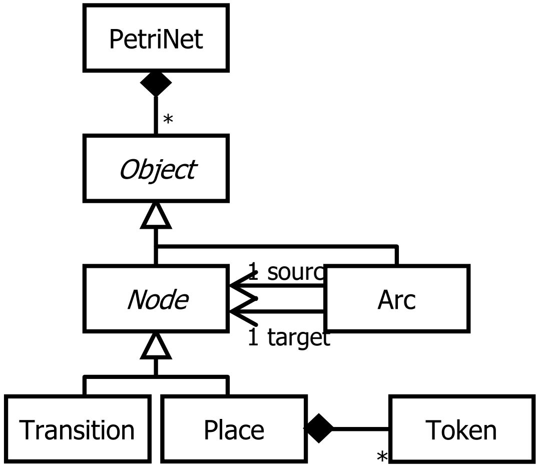
- **Better understanding**
- Communication

- Mapping of instances to XML syntax (XMI)

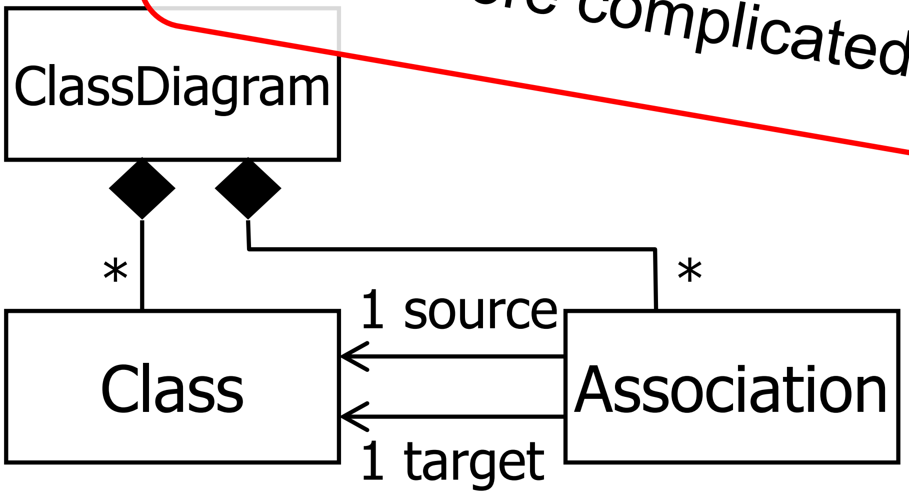
- Automatic code generation
  - API for creating, deleting and modifying model
  - Methods for loading and saving models (in XMI)
  - Standard mechanisms for keeping track of changes (observers)
  - Simple editor (tree editors)

This year, we won't focus on automatic code generation, though!

Class Diagrams are Models too



UML model

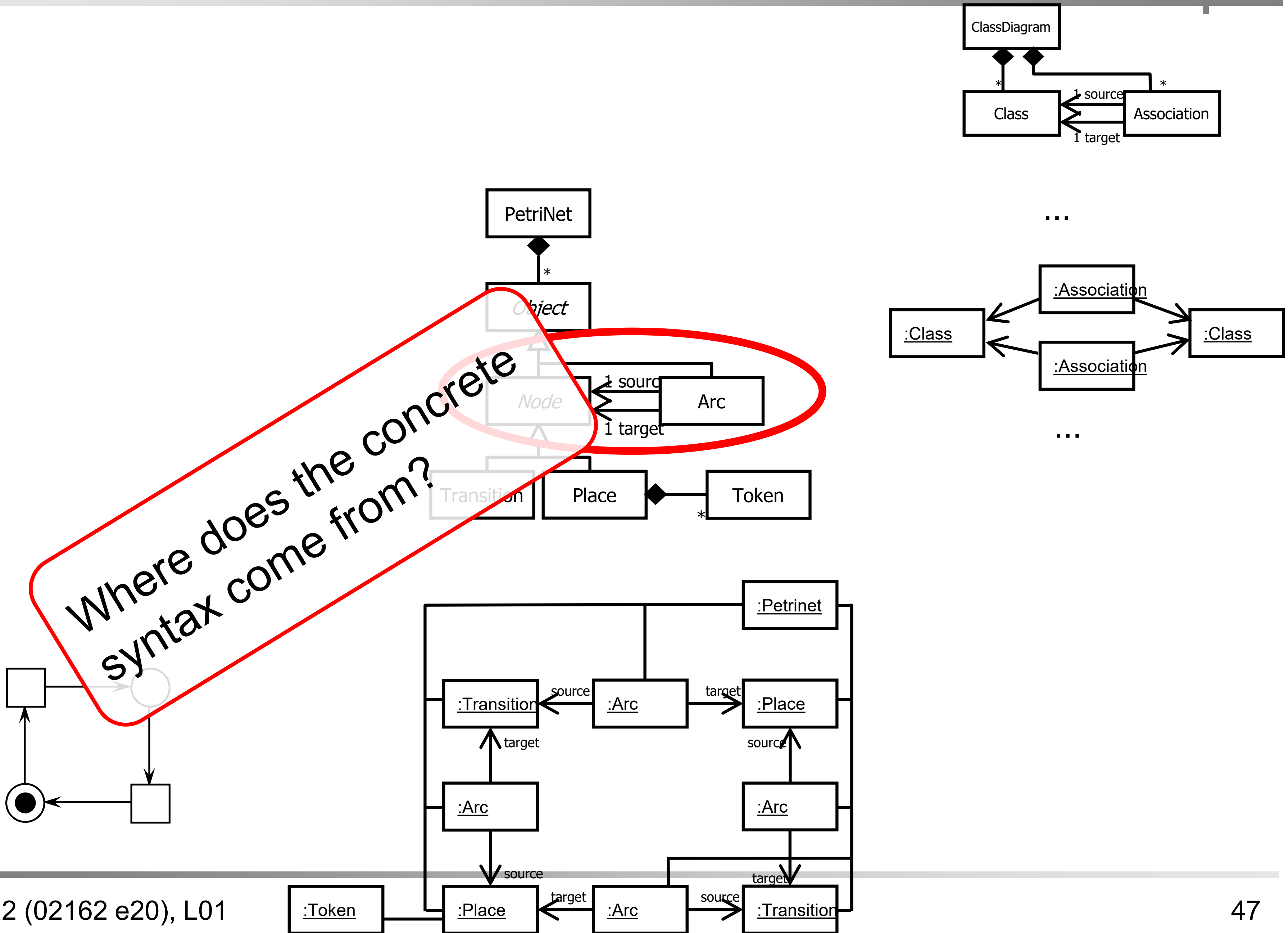


Meta model for UML (class diagrams)

Note: The real model of UML is much more complicated.

The term "meta" model makes more sense now!

Different Meta-levels: MOF



Where does the concrete syntax come from?

Answers:

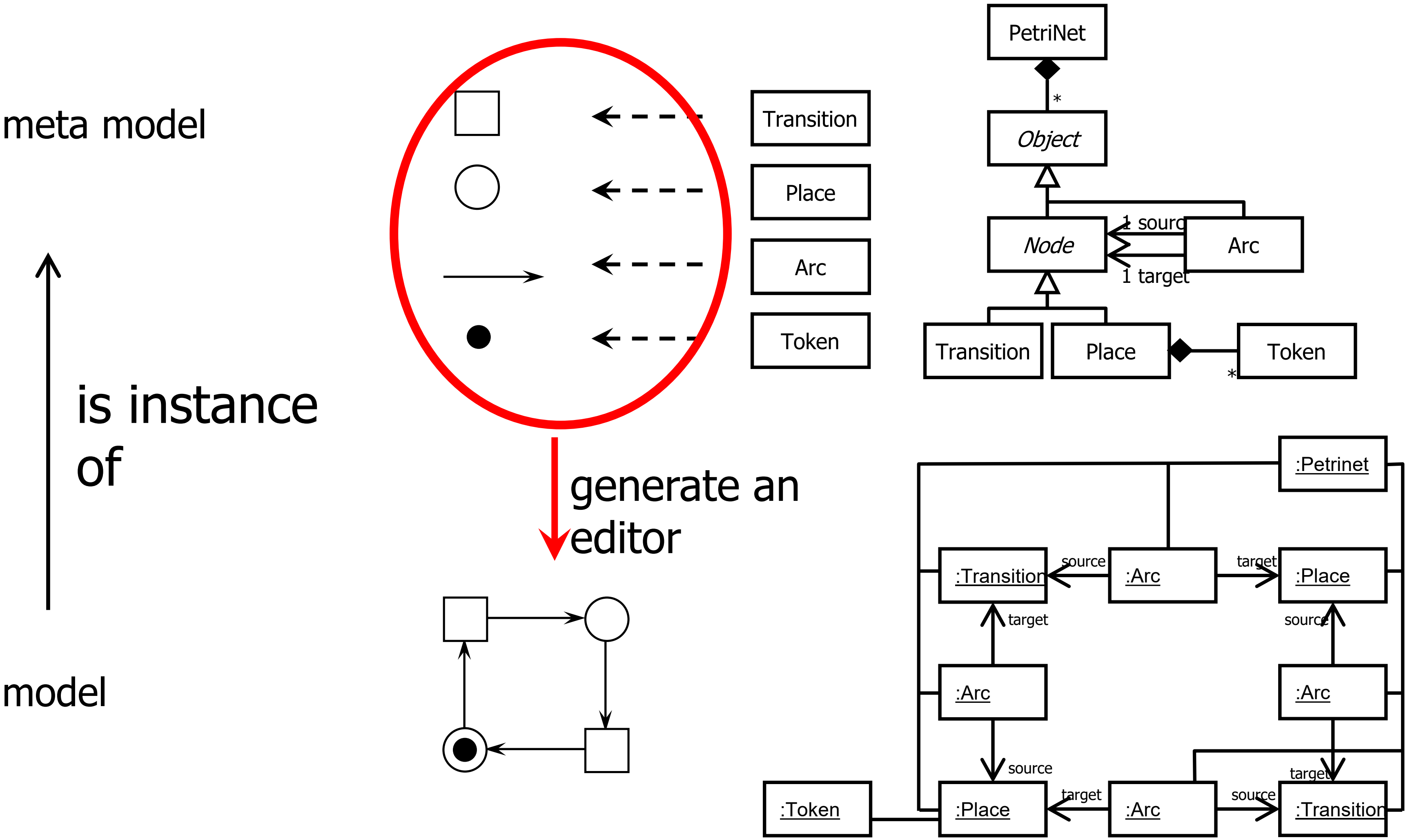
- Program an editor

Not a good answer here!

- Standard technology for mapping abstract to concrete syntax: EMF / GMF / EMFT

Not in the focus of this year's course!

Generation Technologies



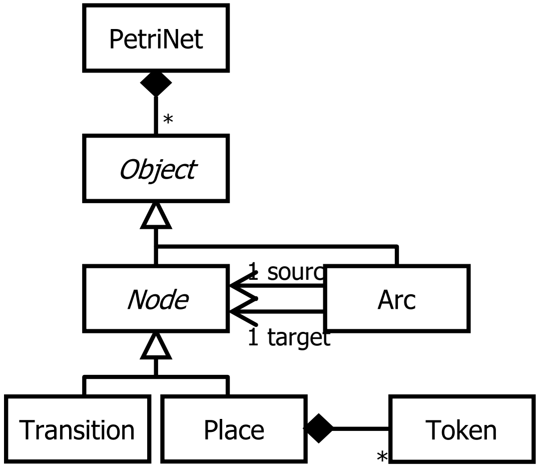
Benefits of Modelling (cntd.)

- Better Understanding
- Mapping of instances to XML syntax (XMI)
- Automatic Code Generation
  - API for creating, deleting and modifying model
  - Methods for loading and saving models (in XMI)
  - Standard mechanisms for keeping track of changes (observers)
  - Editors and GUIs

This year, we won't focus on automatic code generation!

Domain model

- The model we have discussed before has not so much to do with programming (even though code could be generated from it)
- It is about making the concepts of the domain precise: **domain models**



- It is about the "what", not about the "how" (→ slide 13) !

More on this, later in this course!

- Of course, UML class diagrams are also used for modelling the "how": **software models**

Other UML diagrams

In this course, we will use many other kind of UML diagrams, for **different purposes**

- Use cases
- State machines
- Activity diagrams
- Sequence diagrams
- Component diagrams
- ...

UML is a “language”, you learn it only by speaking it!

We will use it as often as we can in the project (and in its documentation).

If you are not familiar with UML, read up on it:

- UML User guide
- UML distilled

Available online as Safari Book or via FindIt (see course’s web page)

Models and Agile

In agile development models are mostly used for informal discussions and **communication**!

Anyway, in order to practice the effective use of models, you will be required to use models and submit models in the documentation in this course.

Documentation is required as part of some releases (and as part of the final submission).