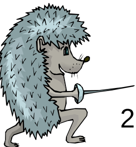
INF-5150 2009
by Øystein Haugen and Ketil Stølen
plus assistant(s) Rayner R. Vintervoll

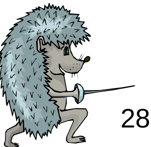
Version 090828

INF 5150



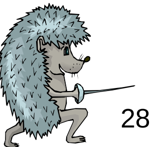
Øystein Haugen <oysteinh@ifi.uio.no>

- 80-81: UiO, Research assistant for Kristen Nygaard
  - 81 : IN 105 together with Bjørn Kirkerud
- 81-84: Norwegian Computing Center, Simula-machine
- 84-88: SimTech, typographical applications
- 88-90: ABB Technology, SDL, prototype SDL tool, ATC
- 89-97: SISU project, methodology, V&V, ITU
- 96-00: Rapporteur ITU for MSC
- 97: Practitioners' verification of SDL systems (dr. scient.)
- 97- 03: Ericsson, NorARC
- 98- 03: Ifi, UiO as Part time Associate Professor
  - IN-TIME (98) IN-RTIMe (99) IN-RTIMe (2000) INFUIT (2001 og 2002)
- 99- : Participates in OMG wrt. UML 2.0
  - Responsible for UML 2.x chapter on Interactions
- 04 - 07 : Associate Professor at Ifi (100%)
- 07- : Senior Researcher at SINTEF ICT
  - Projects on modeling languages e.g. for variability, train control and pay rolls
- 07- : Associate Professor at Ifi (20%)



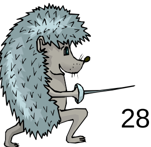
Ketil Stølen <ketil.stolen@sintef.no>

- Leader of Group for Quality and Security Technology at SINTEF
- Professor II at IFI
- Background from University of Manchester (4 years); Technical University Munich (5 years); Institute for Energy Technology (3 years); Norwegian Defence Research Establishment (1 year); SINTEF (9 years)
- PhD in formal methods
- Leading role in the development of the STAIRS method providing the basic foundation for the refinement part of this course
- Leading role in the development of the CORAS method for model-based security analysis providing the basic foundation for the security part of this course
- Is currently managing research projects with a total budget of 35 million NOK

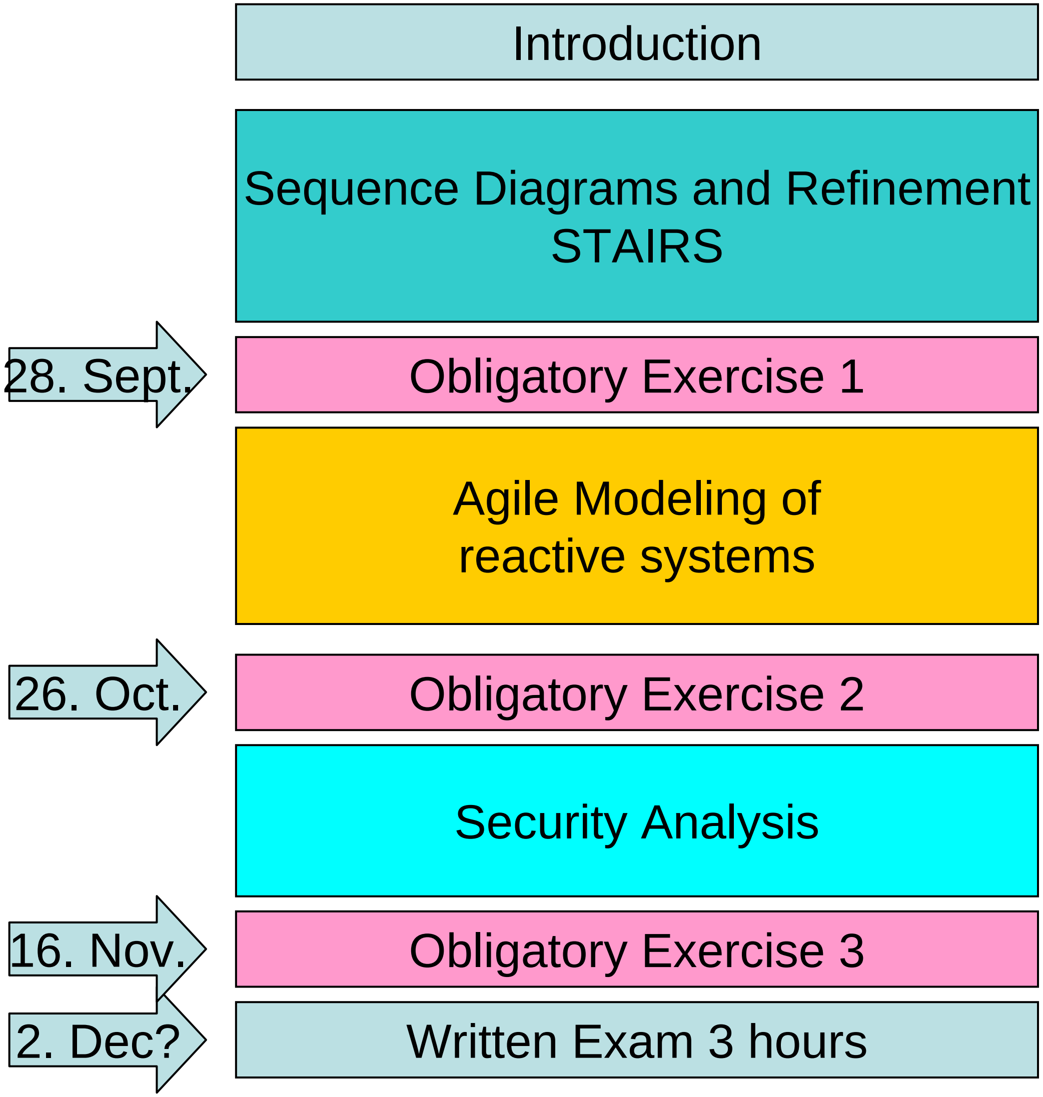


Rayner R. Vintervoll <raynerv@ifi.uio.no>

- Education
  - Bachelor of Informatics, Department of Informatics, University of Oslo
  - Spring 08 semester, School of Information/Department of Sociology, University of California, Berkeley
  - At present: Informatics Master student, Department of Informatics, University of Oslo
- Currently involved with the integration/maintenance of the IFI UML Tool package.
- Took INF5150 Autumn 2007
- Assistant teach INF5150 Autumn 2008



The Course Structure 2009



Introduction

Sequence Diagrams and Refinement
STAIRS

28. Sept. → Obligatory Exercise 1

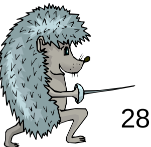
Agile Modeling of
reactive systems

26. Oct. → Obligatory Exercise 2

Security Analysis

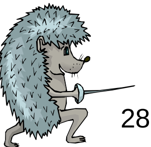
16. Nov. → Obligatory Exercise 3

2. Dec? → Written Exam 3 hours



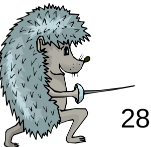
Practical details

- When?
  - Lecture: Friday 9.15 - 12.00 (3B)
  - Exercises: Wednesdays 14.15 – 16.00 (3B)
- Language: English
- Exam
  - Credits: 10 studiepoeng
  - Form: written
  - Grades: A – F or *Bestått / Ikke Bestått* for PhD students (INF9150)
- Obligatory Exercises
  - The obligatory exercises are individual
  - The students may be asked to explain details in their solution



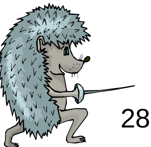
Mandatory Requirements

- Mandatory requirements STAIRS
  - Haugen, Husa, Runde, Stølen: STAIRS towards formal design with sequence diagrams, 2005. SoSyM, Springer Online.
  - Runde, Haugen, Stølen: The Pragmatics of STAIRS, 2006. Springer-Verlag. LNCS 4111.
- Mandatory requirements CORAS
  - den Braber, Hogganvik, Lund, Stølen, Vraasen: Model-based security analysis in seven steps - a guided tour to the CORAS method, 2007. Springer. in BT Technology Journal, pp 101-117.
  - Dahl, Hogganvik, Stølen: Structured semantics for the CORAS security risk modelling language, 2007. SINTEF ICT. Technical Report A970.
- Mandatory requirements UML and modeling
  - Pilone, Dan: UML 2.0 in a Nutshell, 2005. O'Reilly Media. ISBN: 0-596-00795-7.
  - Haugen, Møller-Pedersen, Weigert: Structural Modeling with UML 2.0, 2003. Kluwer. ISBN: 1-4020-7501-4. We have picked out one chapter, but also other chapters are interesting.
- The lecture slides are mandatory requirements
- Your own solutions to the obligatory exercises are also mandatory requirements



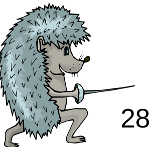
INF5150: Unassailable IT Systems (BZZZ)

- The title of the course is probably not intuitive?
- What are your expectations?
  - Discuss with your neighbor to come up with
  - 3 explicit expected goals for your participation in this course
    - what you expect to learn
    - what efforts you expect to put into it
    - what you expect to avoid
    - special requirements?
- Spend 2 minutes on this!
- ... and then we shall record your expectations



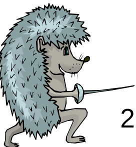
Goal: Unassailable IT-Systems

- The course INF-UIT aims at teaching the students
  - how software is made unassailable meaning that
    - the software is easily analyzed with respect to reliability and dependability
    - the software is easily maintained
- The overall goal is to explain
  - how practical software development can benefit from theories about
    - state machines
    - refinement
    - formal reasoning
    - modularity
    - security and related matters



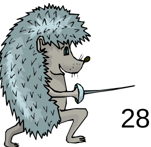
Unassailable IT-Systems

- Unassailable?
- IT?
- Systems?



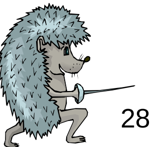
Unassailable

- Not assailable : not liable to doubt, attack, or question
- Where is this important?
  - for all software?
    - to some extent, but possibly less than one would like to think
  - for some critical software
    - telecom
    - surveillance (of patients, of production processes)
    - within computers themselves
- This course is not concerned with attacks that come from hackers towards data bases with sensitive content
  - we are concerned with helping software to perform properly even in unexpected situations



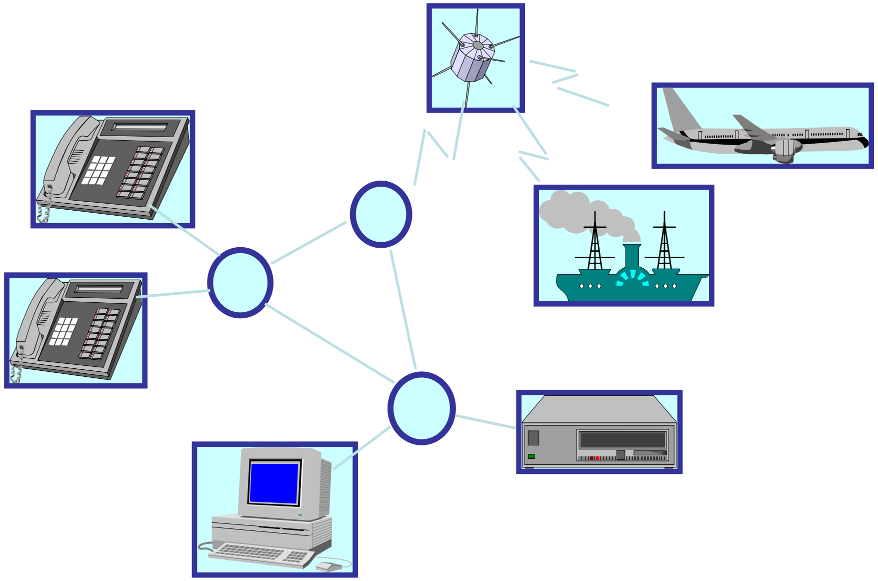
IT?

- Information Technology
  - using computers
  - with emphasis on practical systems
  - with emphasis on behavior
- Engineering
  - Well acknowledged and asserted techniques
  - Creativity only when and where needed
  - Replication of earlier efforts
  - Pragmatics as well as theory

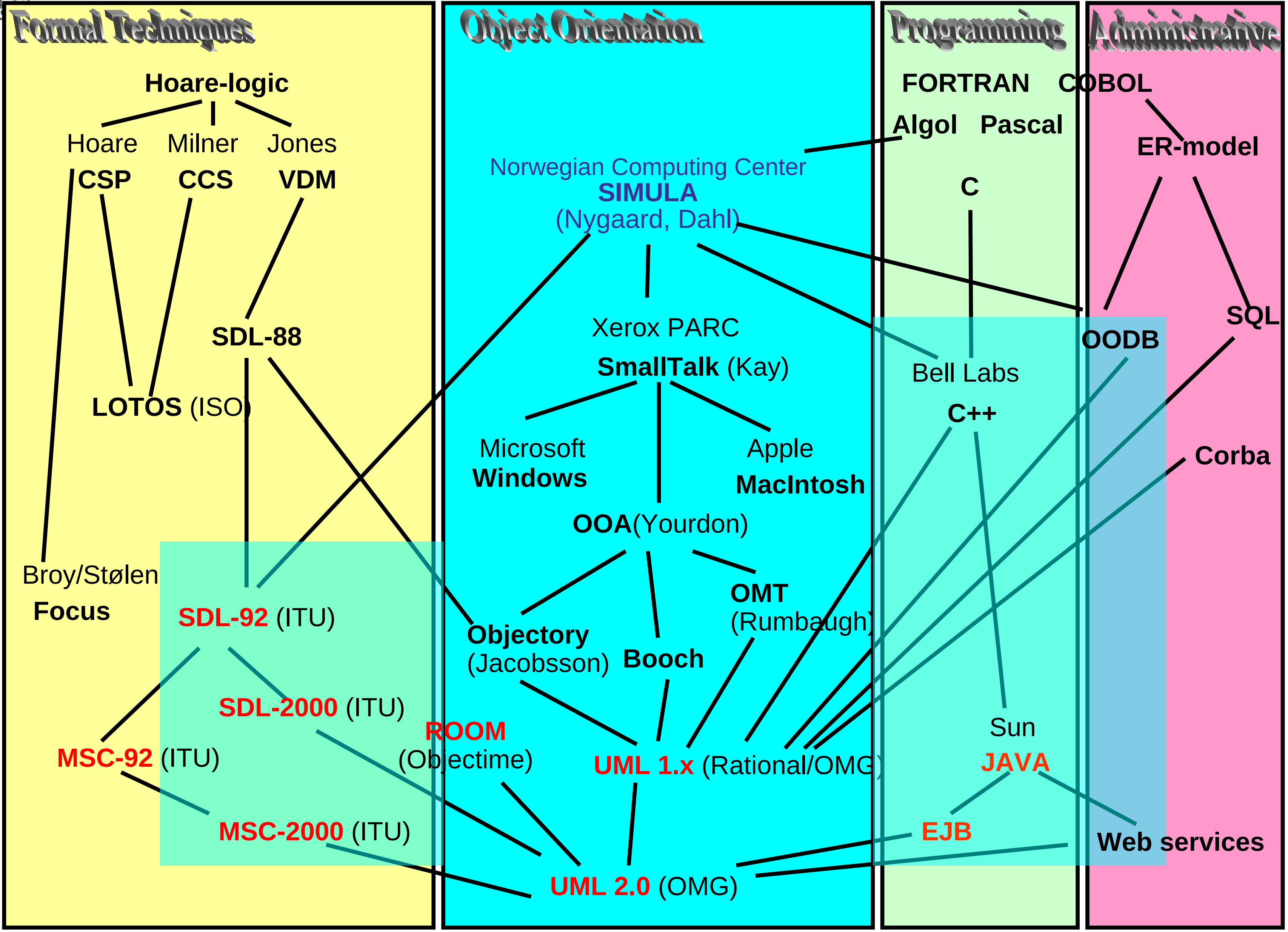


Systems?

- distributed
- concurrent
- real-time
  - In synchrony with real life
  - often small amounts of time for each service e.g. Automatic Train Control
  - the actual durations may or may not be significant
- reactive
- heterogeneous
- complex



UML 2.0 in the history of languages



Formal Techniques

Hoare-logic

Hoare CSP

Milner CCS

Jones VDM

SDL-88

LOTOS (ISO)

Broy/Stølen Focus

SDL-92 (ITU)

SDL-2000 (ITU)

MSC-92 (ITU)

MSC-2000 (ITU)

Object Orientation

Norwegian Computing Center SIMULA (Nygaard, Dahl)

Xerox PARC SmallTalk (Kay)

Microsoft Windows

Apple MacIntosh

OOA(Yourdon)

Objectory (Jacobsson)

Booch

OMT (Rumbaugh)

ROOM (Objectime)

UML 1.x (Rational/OMG)

UML 2.0 (OMG)

Programming

FORTRAN

Algol

Pascal

C

Bell Labs C++

Sun JAVA

EJB

Administrative

COBOL

ER-model

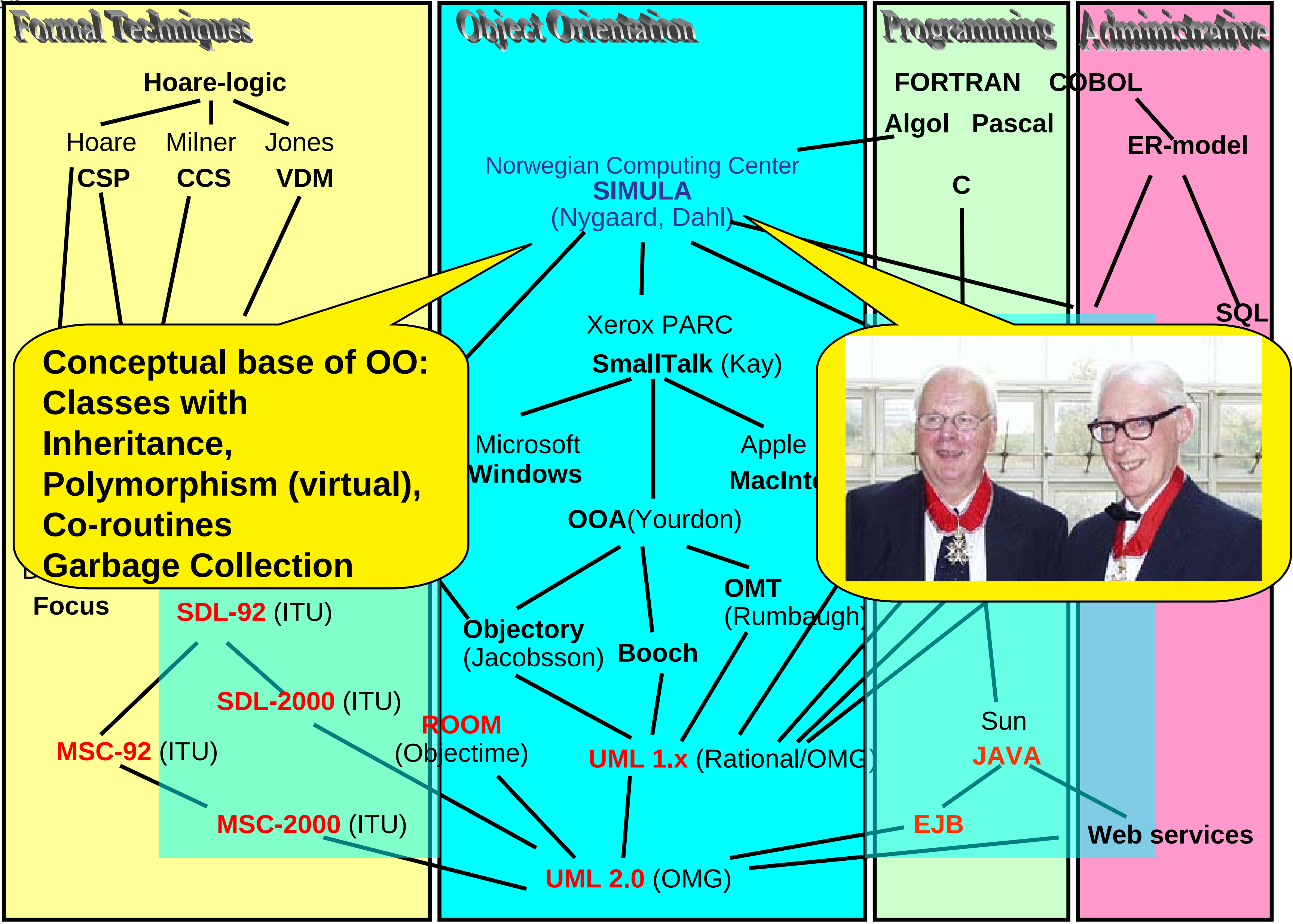
OODB

SQL

Corba

Web services

The founding fathers



Formal Techniques

Hoare-logic

Hoare Milner Jones

CSP **CCS** **VDM**

D

Focus

SDL-92 (ITU)

SDL-2000 (ITU)

MSC-92 (ITU)

MSC-2000 (ITU)

Object Orientation

Norwegian Computing Center

SIMULA

(Nygaard, Dahl)

Xerox PARC

SmallTalk (Kay)

Microsoft

Windows

Apple

MacInt

OOA(Yourdon)

Objectory

(Jacobsson)

Booch

OMT

(Rumbaugh)

ROOM

(Objectime)

UML 1.x (Rational/OMG)

UML 2.0 (OMG)

Programming

FORTRAN **COBOL**

Algol **Pascal**

C

Sun

JAVA

EJB

Administrative

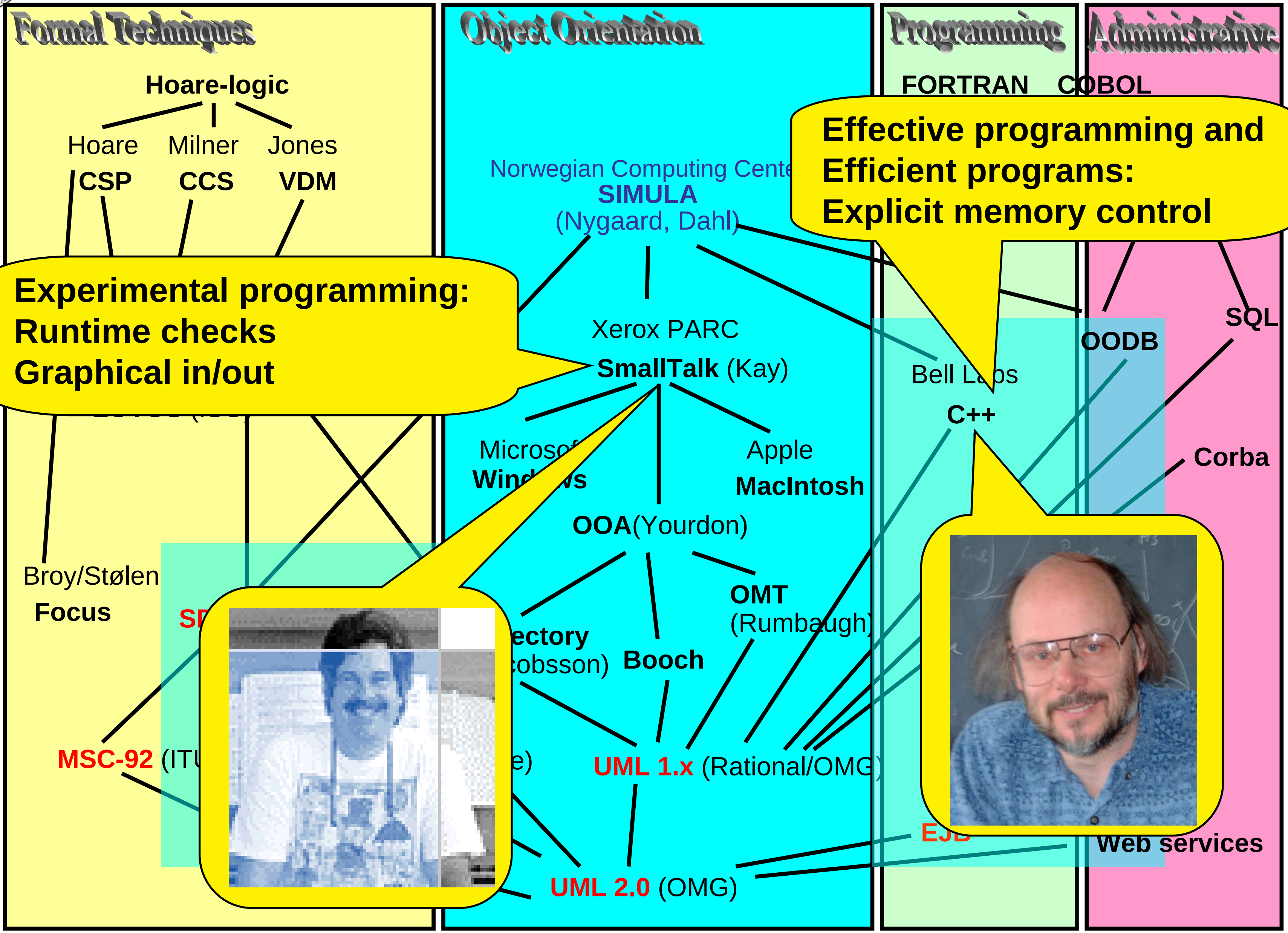
ER-model

SQL

Web services

Conceptual base of OO:
Classes with Inheritance,
Polymorphism (virtual),
Co-routines
Garbage Collection

Making OO Popular and Commercial



Formal Techniques

Hoare-logic

Hoare CSP

Milner CCS

Jones VDM

Broy/Stølen Focus

SD

MSC-92 (ITU

e)

Object Orientation

Norwegian Computing Cente

SIMULA

(Nygaard, Dahl)

Xerox PARC

SmallTalk (Kay)

Microsof

Wind ws

Apple

MacIntosh

OOA(Yourdon)

ectory

cobsson)

Booch

OMT

(Rumbaugh)

UML 1.x (Rational/OMG)

UML 2.0 (OMG)

Programming

FORTRAN

Bell La ps

C++

EJB

Administrative

COBOL

OODB

SQL

Corba

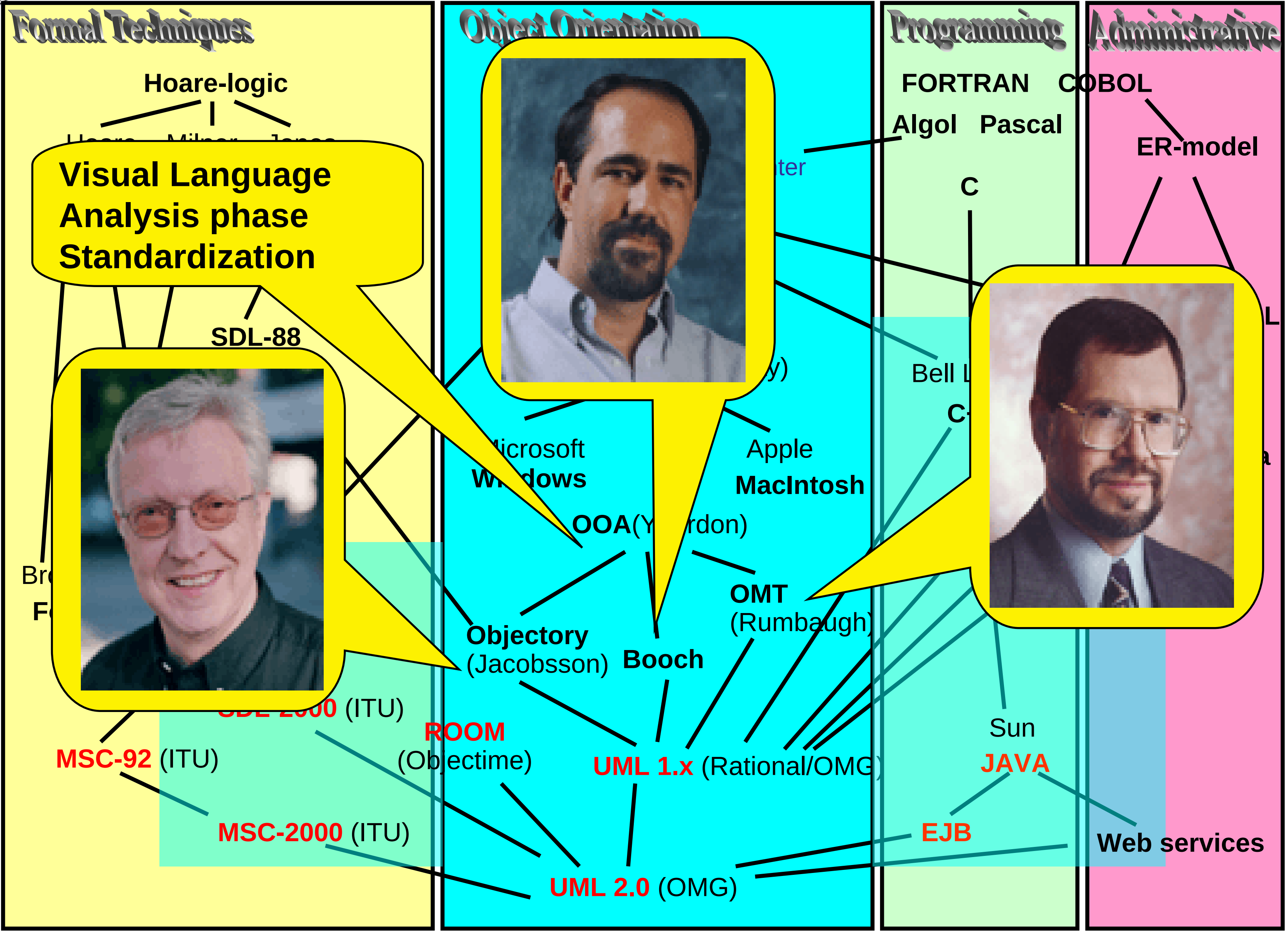
Web services

Experimental programming:
Runtime checks
Graphical in/out

Effective programming and
Efficient programs:
Explicit memory control

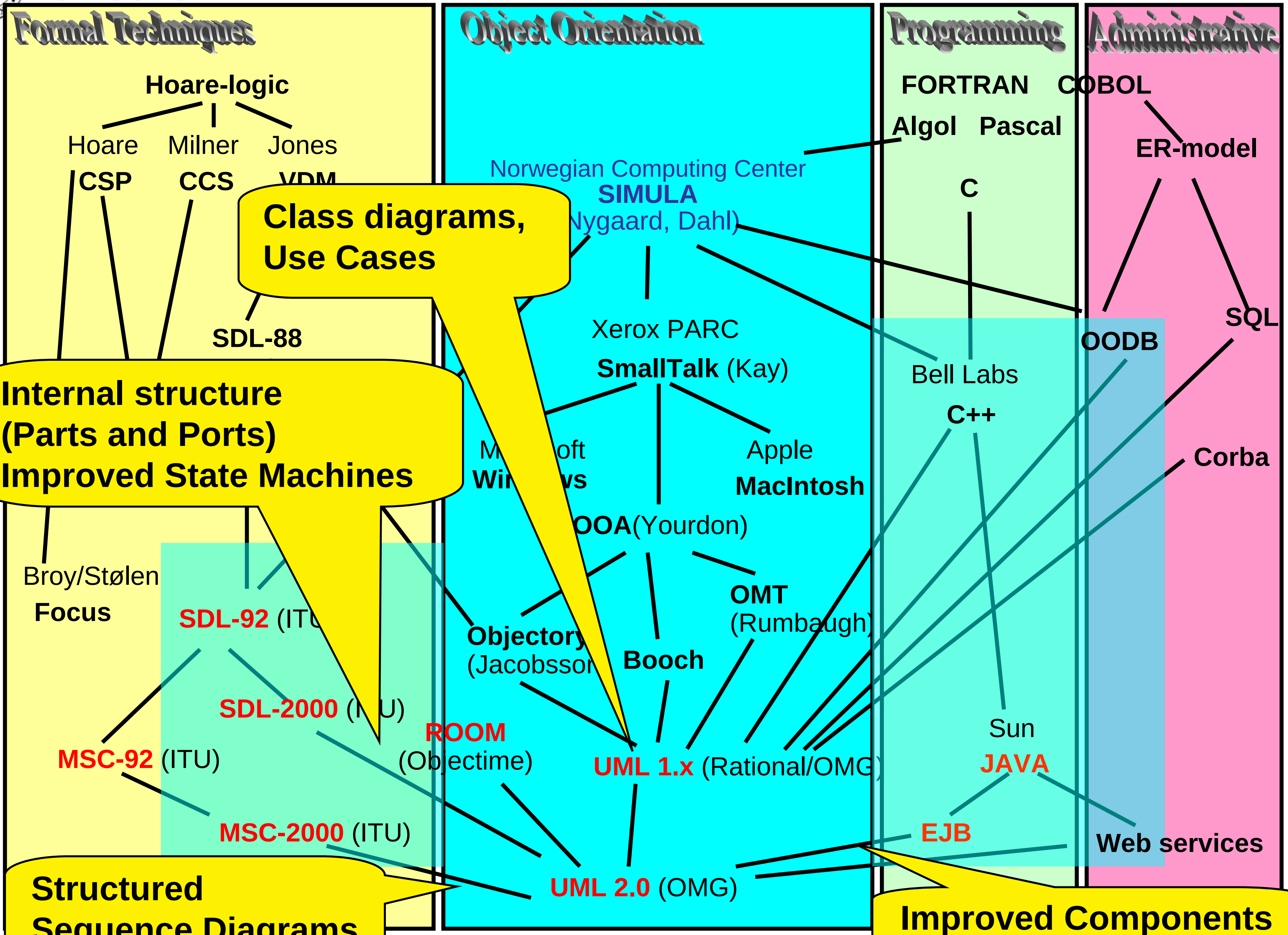
INF 5150

The Three Amigos



Visual Language
Analysis phase
Standardization

Influences on UML 2.0



Formal Techniques

Hoare-logic

Hoare Milner Jones

CSP CCS VDM

SDL-88

Broy/Stølen

Focus

SDL-92 (ITU)

SDL-2000 (ITU)

MSC-92 (ITU)

MSC-2000 (ITU)

Object Orientation

Norwegian Computing Center

SIMULA

(Nygaard, Dahl)

Xerox PARC

SmallTalk (Kay)

Microsoft

Windows

Apple

MacIntosh

OOA(Yourdon)

Objectory

(Jacobsson)

Booch

OMT

(Rumbaugh)

ROOM

(Objectime)

UML 1.x (Rational/OMG)

UML 2.0 (OMG)

Programming

FORTRAN

Algol Pascal

C

Bell Labs

C++

Sun

JAVA

EJB

Administrative

COBOL

ER-model

SQL

OODB

Corba

Web services

Class diagrams,
Use Cases

Internal structure
(Parts and Ports)
Improved State Machines

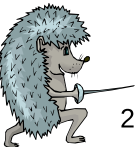
Structured
Sequence Diagrams

Improved Components

INF 5150

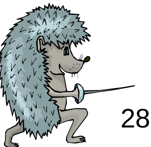
What language(s) to use? Why? (BZZZ)

- Requirements
  - used in practice for real engineering
  - expressive
  - visual
  - precise
  - trendy
- Alternatives?
  - java (Sun)
    - possibly supplied with selected libraries
  - SDL (ITU)
  - MSC (ITU)
  - UML 1.x (OMG)
  - UML 2.0 (OMG)



Why choosing UML 2?

- Pro
  - UML is definitely trendy wrt. modeling languages
  - UML is standardized by open standardization organization (OMG)
  - UML 2.0 has most features of MSC and SDL
  - UML 2.0 is more precise and executable than UML 1.x
  - UML 2.0 is supported by more than one tool, and can be expressed through any drawing tool like Powerpoint, Visio, Framemaker
  - UML 2.0 is now, UML 1.x is history soon
- Con
  - Good UML 2 tools are hard to find
  - Real programmers do not use modeling languages anyway



UML Diagrams

- UML diagrams:
  - Use case diagram
  - Static structure diagrams:
    - Class / object diagram
    - Collaboration
    - Composite structure diagram
  - Behavior diagrams:
    - Sequence diagram
    - Communication diagram
    - State diagram
    - Activity diagram
  - Implementation diagrams:
    - Component diagram
    - Deployment diagram

Use:

Identifying main system functions

Domain and application modeling

internal structure of objects

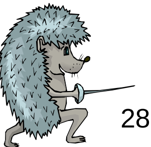
Interactions between objects

Class behaviour (state oriented)

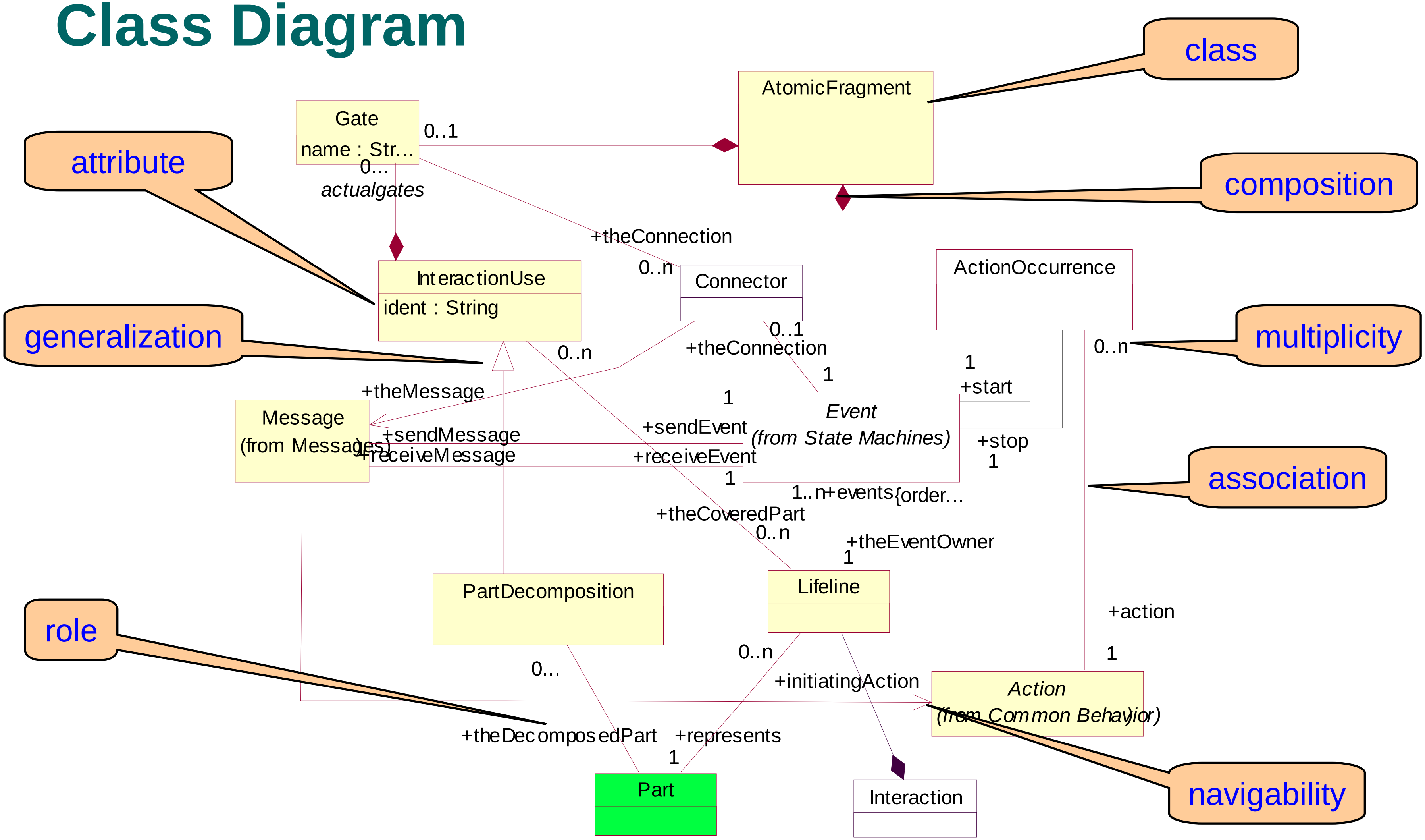
Ditto (action oriented)

For software structure

For hardware/software structure



Class Diagram



class

AtomicFragment

Gate

name : Str...

0..1

0...

actualgates

attribute

composition

+theConnection

0..n

InteractionUse

ident : String

Connector

ActionOccurrence

generalization

0..1

+theConnection

0..n

multiplicity

1

+start

1

+theMessage

0..n

1

Message
(from Messages)

+sendMessage

+receiveMessage

+sendEvent

+receiveEvent

1

Event
(from State Machines)

+stop

1

1..n+events{order...

association

+theCoveredPart

0..n

+theEventOwner

1

PartDecomposition

Lifeline

+action

role

1

0...

0..n

+initiatingAction

Action
(from Common Behavior)

+theDecomposedPart

+represents

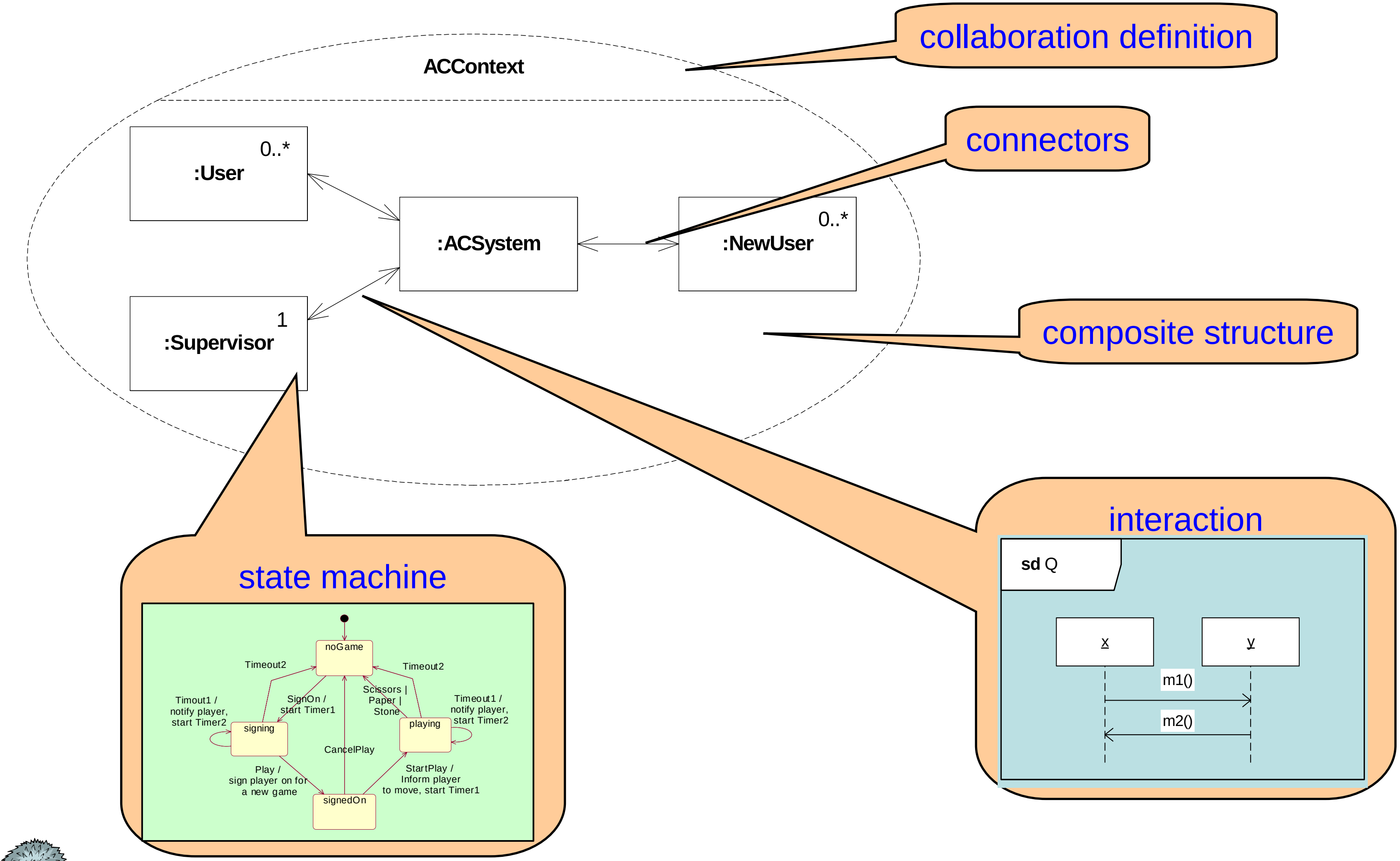
1

Part

Interaction

navigability

Composite Structure



collaboration definition

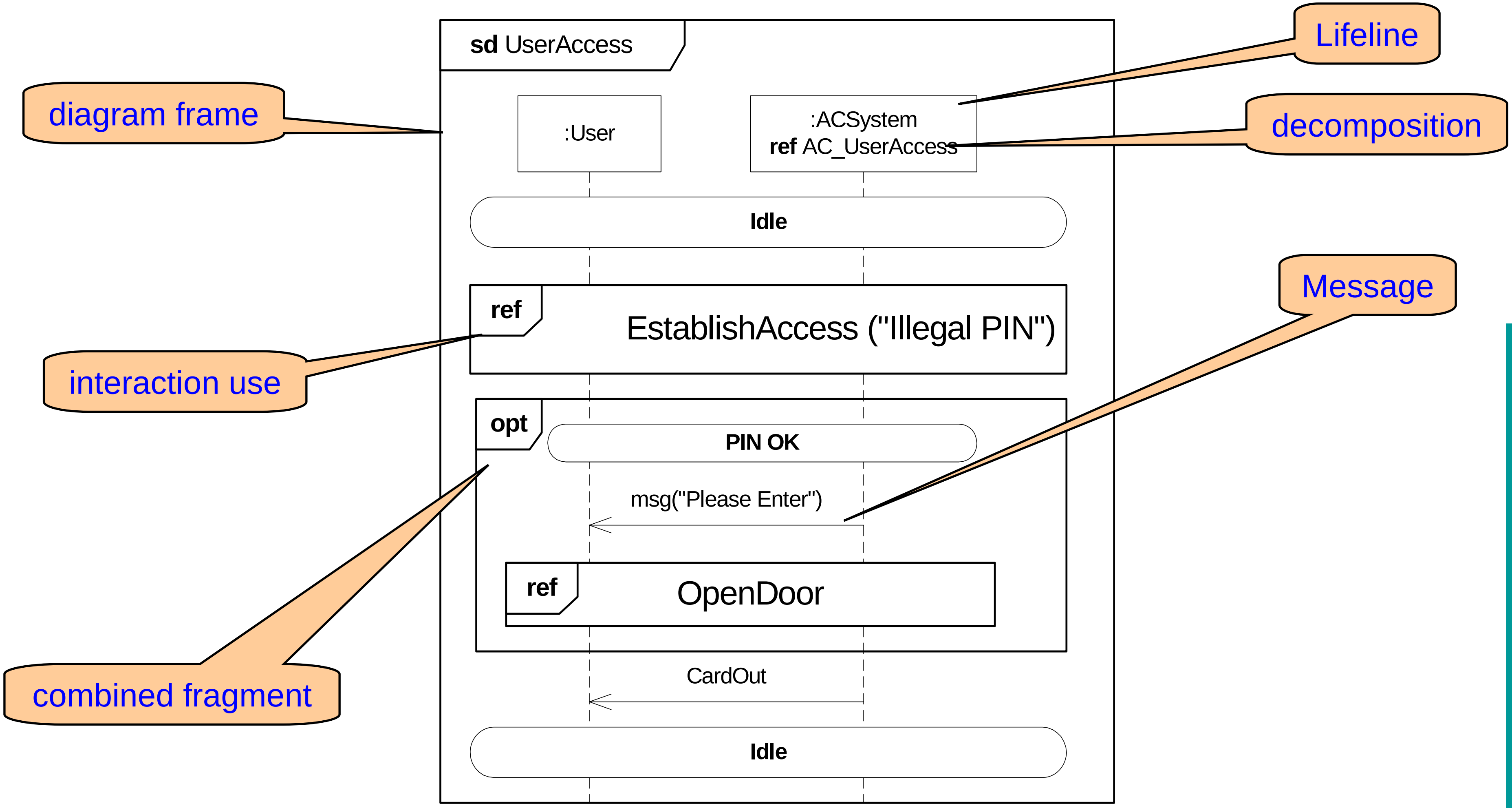
connectors

composite structure

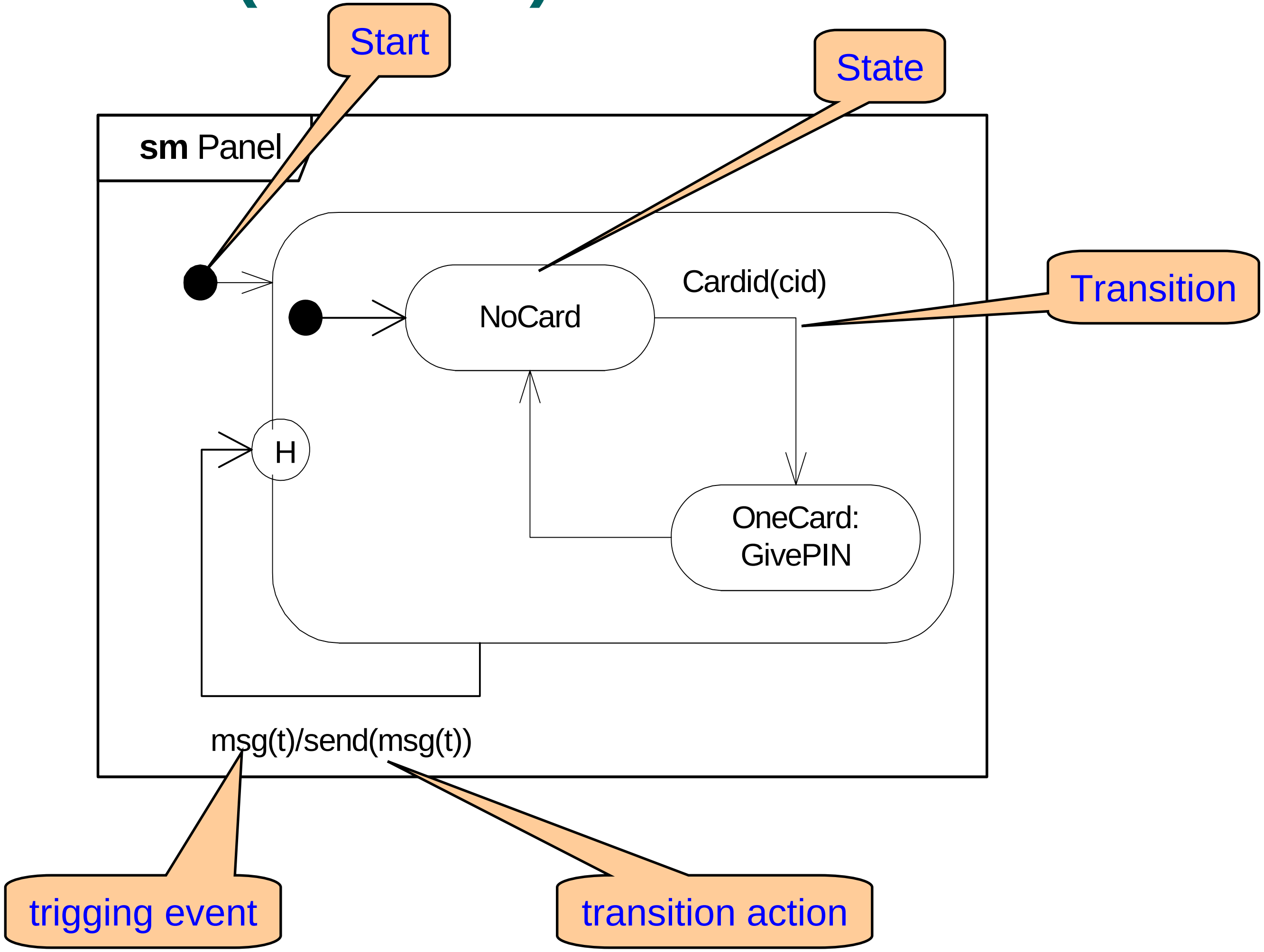
state machine

interaction

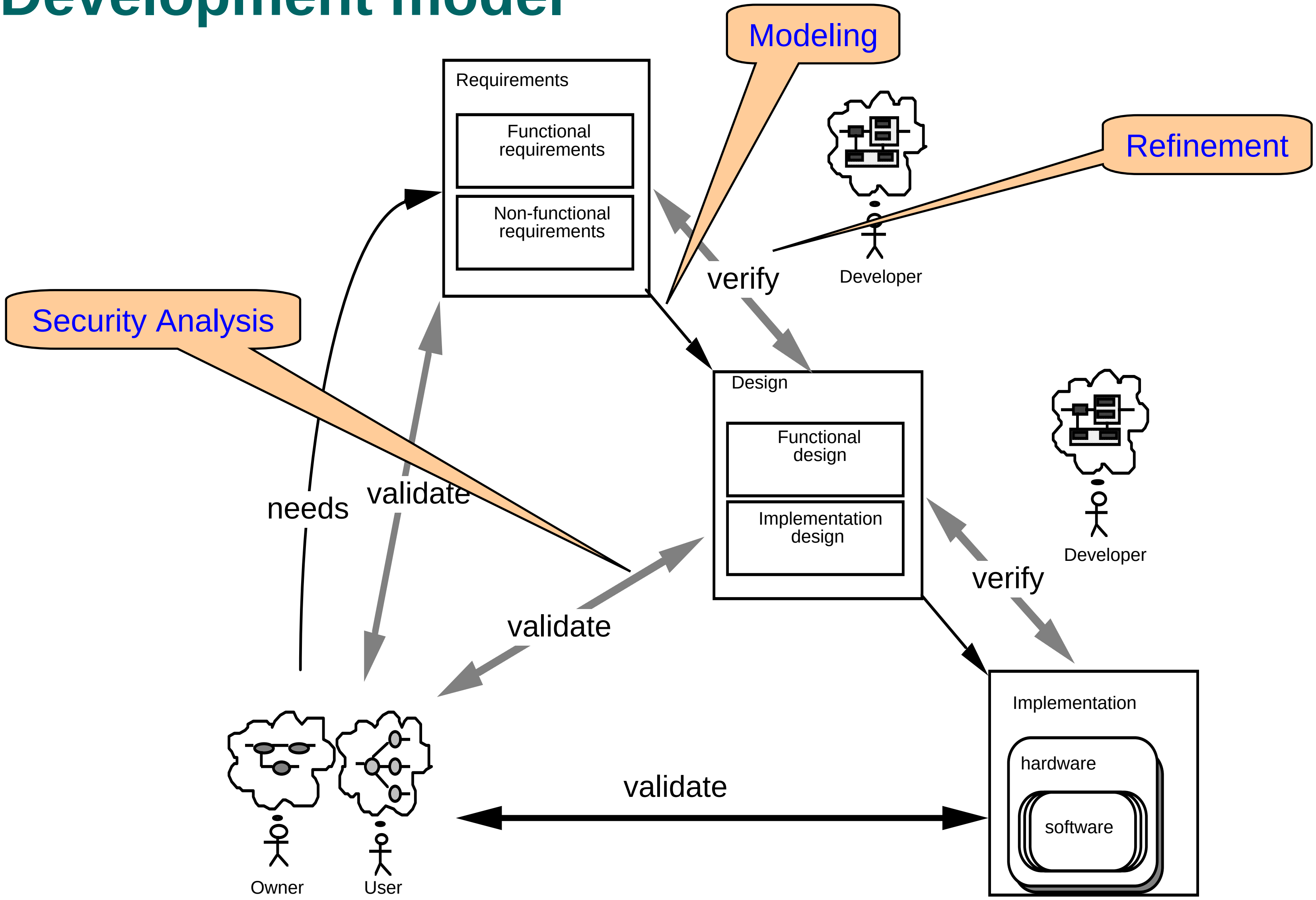
Sequence Diagram (UML 2)



State Machines (UML 2.0)



Development model



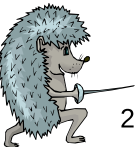
STAIRS – Steps To Analyze Interactions with Refinement Semantics

RELAIS GOURMANDS
supplementing
narrowing
KOSHER DELI
detailing

Thanks to Microsoft clipart and Restaurant Bagatelle’s web-site

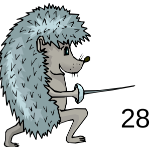
Refinement

- Refine = to free (as metal, sugar, or oil) from impurities or unwanted material
  - here: to make more exact, to reduce the set of legal solutions
  - in particular: to reduce the set of legal histories
- The role of histories
  - Histories model system runs
  - Specifications are modeled by sets of histories
- The need for a precise semantics
  - Syntax, Semantics, Pragmatics
- The assumption/guarantee paradigm
  - The assumption describes the properties of the environment in which the specified component is supposed to run
  - The guarantee characterizes the constraints that the specified component is required to fulfill whenever the specified component is executed in an environment that satisfies the assumption



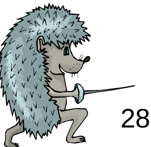
Three main notions of refinement

- Property refinement
  - *requirements engineering:* requirements are added to the specification in the order they are captured and formalized
  - *incremental development:* requirements are designed and implemented in a step-wise incremental manner
- Interface refinement
  - *type implementation:* introducing more implementation-dependent data types
  - *change of granularity:* replacing one step of interaction by several, or the other way around
- Conditional refinement
  - *imposing boundedness:* replacing unbounded resources by implementable bounded resources
  - *change of protocol:* replacing abstract communication protocols by more implementation-oriented communication protocols



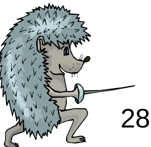
Objectives for the lectures on refinement

- The lectures on refinement will
  - motivate and explain the basic instruments and principles for defining notions of refinement
    - this includes
      - using histories to model executions
      - the notion of an observer
      - understanding the assumption/guarantee principle
  - explain the following refinement concepts in a UML setting
    - property refinement
    - interface refinement
    - conditional refinement
  - demonstrate refinement in examples
- The exercises on refinement will
  - train you in the art of refining, and prepare you for the exam



Modeling: How important are languages?

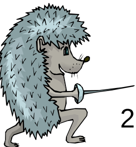
- Not very important
  - “Syntactic sugar”
- Very important
  - “Understanding through describing”



Modeling Methodology

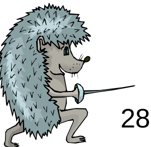
- A good language helps a lot
  - but is hardly sufficient
  - you need to know how to use the language also
- A good method is hard to find
  - easy to understand
  - easy to believe in
  - easy to follow
  - easy to modify
  - easy to get positive effects

  - easy to cheat?
  - easy to overlook?
  - easy to misuse?
  - hard to evaluate?



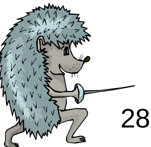
”Agile modeling”

- ”agile”
  - = having a quick resourceful and adaptable character
- executable models!
- very stepwise approach
  - each step will have its specification and executable model
  - each step should be tested
- We shall use one example throughout the course
  - with many steps
  - intended to be mirrored by the project exercise model
- Every week a working program!



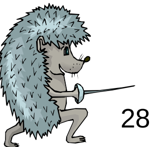
Manifesto for Agile Software Development

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



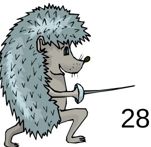
Dialectic Software Development

- Software Development is a process of learning
  - once you have totally understood the system you are building, it is done
- Learning is best achieved through conflict, not harmony
  - discussions reveal problematic points
  - silence hides critical errors
- By applying different perspectives to the system to be designed
  - inconsistencies may appear
  - and they must be harmonized
- Inconsistencies are not always errors!
  - difference of opinion
  - difference of understanding
  - misunderstanding each other
  - a result of partial knowledge
- Reliable systems are those that have already met challenges



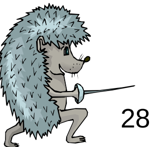
Verification and Validation

- Barry Boehm, 1981:
  - Verification: To establish the truth of correspondence between a software product and its specification (from the Latin veritas, “truth”).
    Are we building the product right?
  - Validation: To establish the fitness or worth of a software product for its operational mission (from the Latin valere, “to be worth”).
    Are we building the right product?
- Quality
  - process quality **= meeting the specification**
  - system quality **= playing the role required by the environment.**
- Quality assurance
  - Constructive **methods that aim to generate the right results in the first place**
  - Corrective **methods that aim to detect errors and make corrections.**

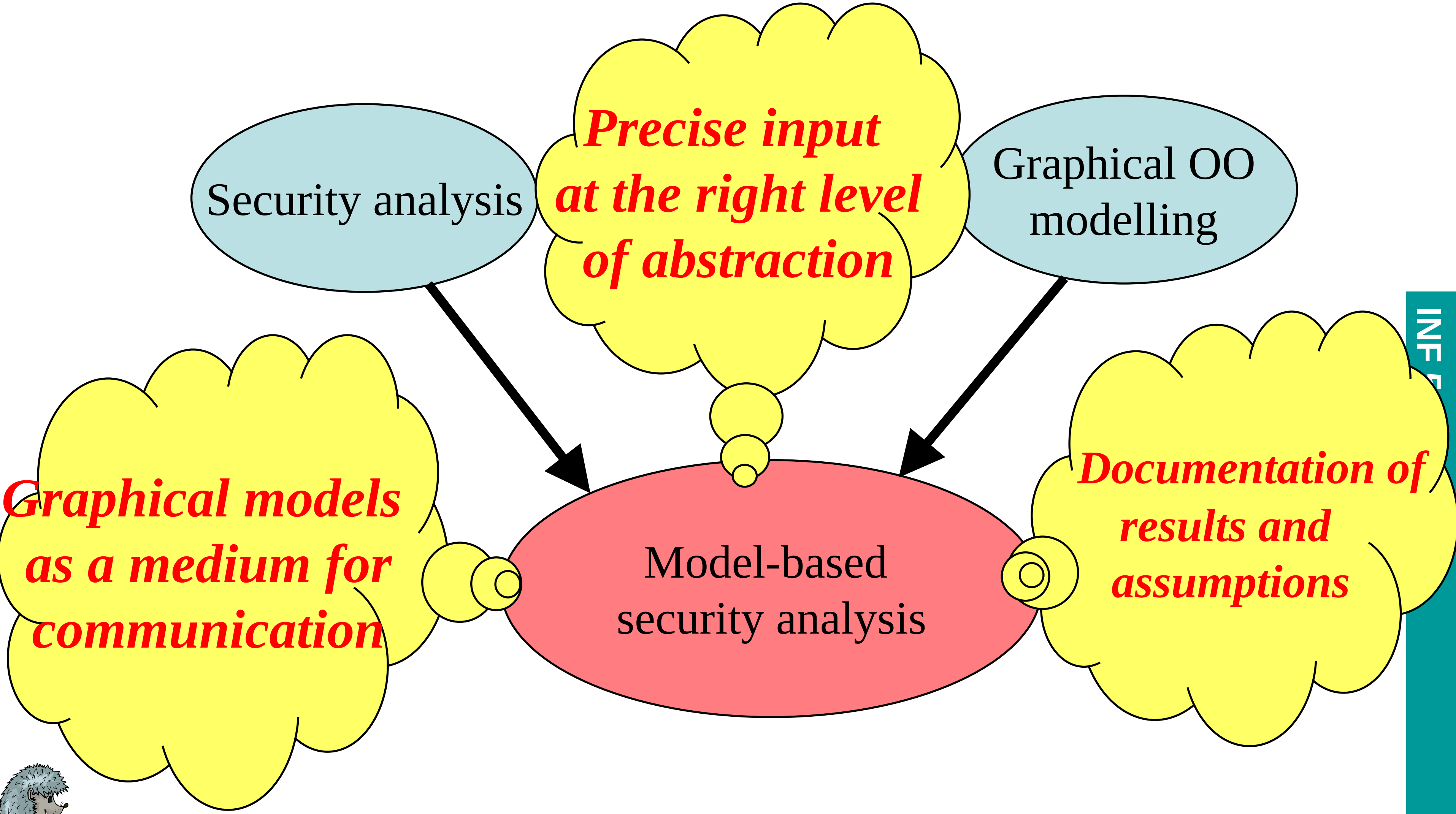


Model-based security analysis

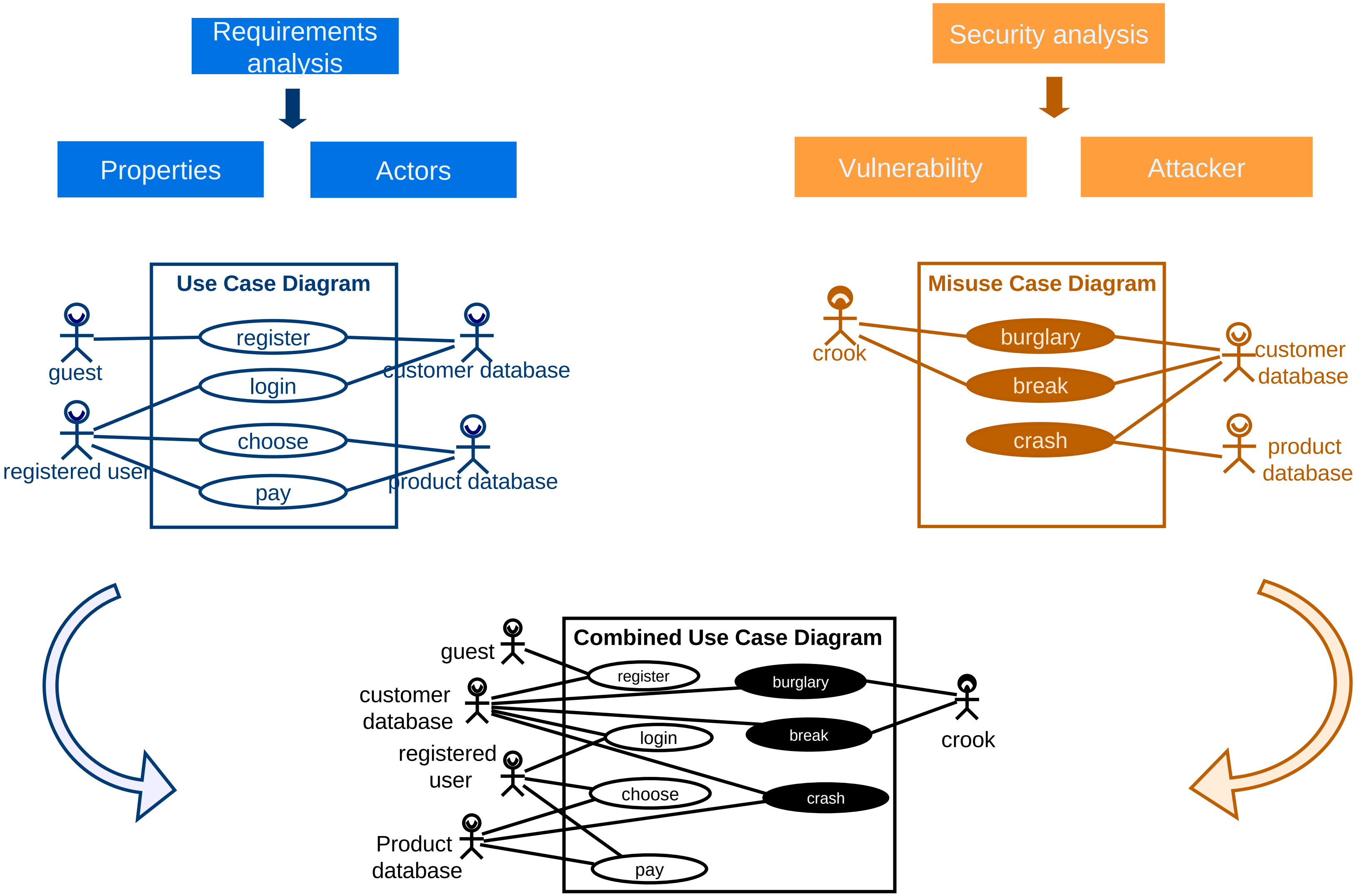
- Risk analysis is a systematic use of available information to
  - determine how often specified events may occur
  - the magnitude of their consequences
- Model-based security analysis is the tight integration of state-of-the art modeling methodology in the security risk analysis process
- Model-based security analysis is motivated by
  - Precision improves the quality of security analysis results
  - Graphical UML-like diagrams are well-suited as a medium for communication between stakeholders involved in a security analysis; the danger of throwing away time and resources on misconceptions is reduced
  - The need to formalize the assumptions on which the analysis depends; this reduces maintenance costs by increasing the possibilities for reuse
  - Provides a basis for tight integration of security analysis in the system development process; this may considerably reduce development costs since undesirable solutions are weeded out at an early stage



Three dimensions of model-based security analysis


Precise input at the right level of abstraction
Security analysis
Graphical OO modelling
Graphical models as a medium for communication
Model-based security analysis
Documentation of results and assumptions

Requirements analysis versus security analysis



Requirements analysis

Properties

Actors

Security analysis

Vulnerability

Attacker

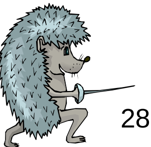
Use Case Diagram

Misuse Case Diagram

Combined Use Case Diagram

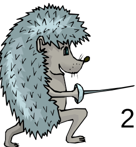
Objectives for the lectures on security analysis

- classify notions of dependability
- introduce, motivate and explain the basic notions and principles for risk management in general and security risk analysis in particular
- relate risk management to system development
- describe the various processes involved in risk management
- motivate and illustrate model-based security analysis
- demonstrate the usage of concrete analysis methodology



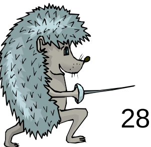
Obligatory Exercises

- Oblig 1
  - will be on refinement
  - based on a given basic model described by sequence diagrams
- Oblig 2
  - Executable models!
  - Test specifications
- Oblig 3
  - Security analysis



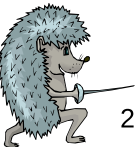
UML Modeling Tool

- IBM Rational RSM 7.5
  - based on Eclipse 3.4
  - Commercial tool, free for you
- Sequence Diagram editor (SeDi) plugin
  - the best sequence diagram editor there is (*Andreas Limyr, Frank Davidsen, Rayner R. Vintervoll,,*)
  - tightly integrated with RSM – works on the same repository
- UML to JavaFrame transformer
  - push a button – executable UML! (*Asbjørn Willersrud*)
- JFDebug
  - model-oriented debugger (*Jonas Winje*)
- Consistency Checker (*Bjørn Brændshøi*)
  - consistency between Interactions and State Machines



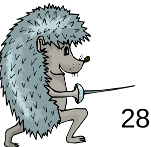
RSM IFI UML – challenges and upsides

- RSM is a commercial tool
  - PRO: maintained, reasonable quality
  - CON: limited insight into the details of the tool
  - CON: the students cannot use it for free outside studies
- RSM IFI UML comprises IFI-made plugins
  - made by Master students
  - used by Master and Bachelor students for years
  - cutting edge technology
    - with astonishing functionality
    - and possibly some irritating bugs

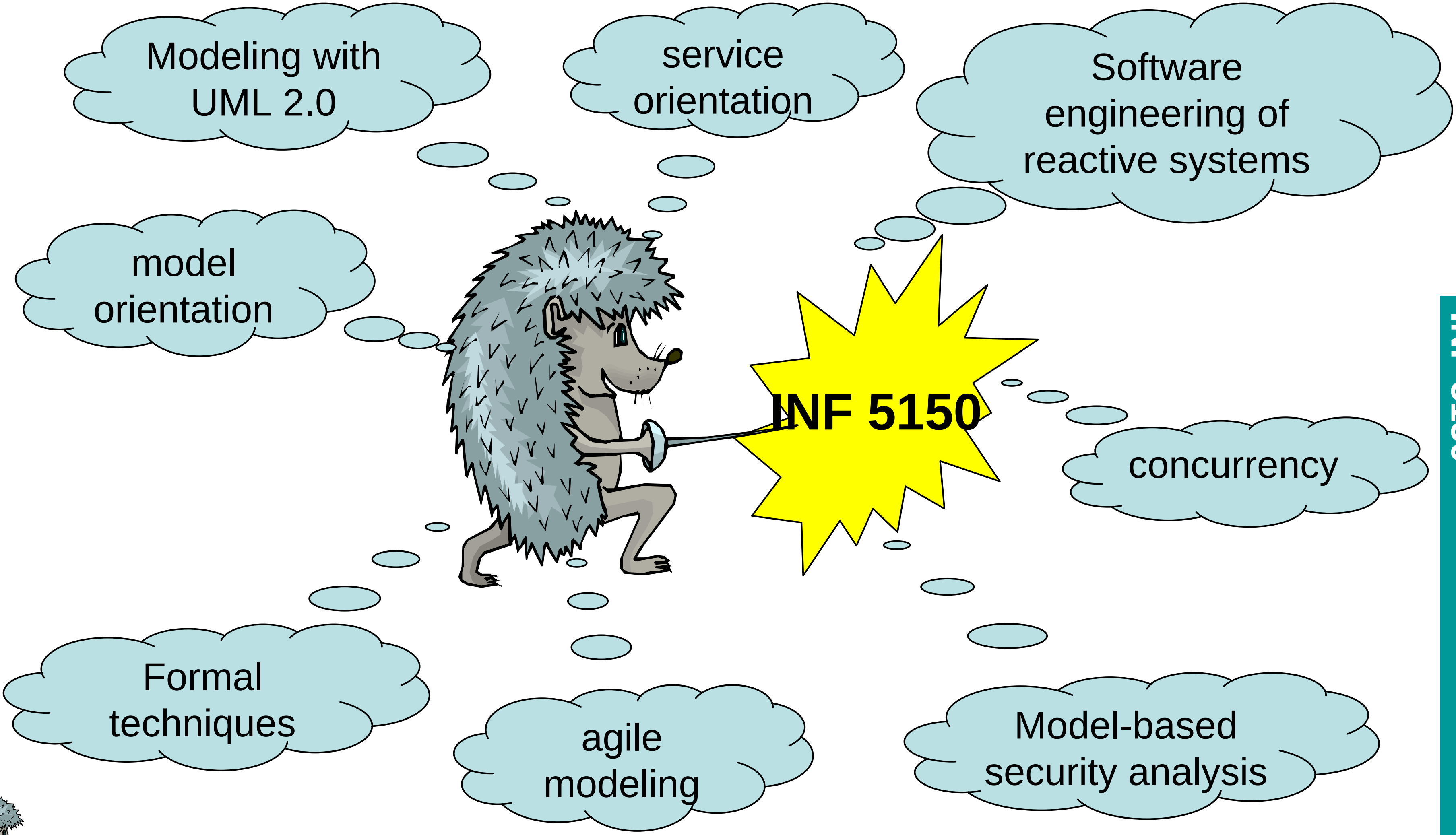


CORAS Risk Analysis Tool

- The CORAS-tool available as open source (LGPL-license):
  - http://coras.sourceforge.net/
- Based on other open software (Apache Cocoon, eXist XML database)
- Created by SINTEF



INF 5150 and the buzzwords



INF 5150

Modeling with UML 2.0

service orientation

Software engineering of reactive systems

model orientation

concurrency

Formal techniques

agile modeling

Model-based security analysis