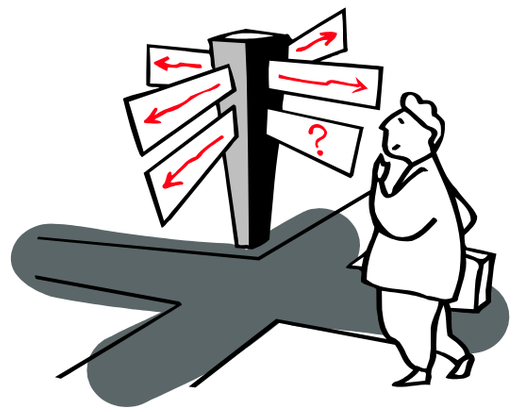
CHAPTER 1 – Introduction



Software Engineering

- Why & What
- Product & Process

  ⇒ Correctness & Traceability

Software Process

- Activities
- Waterfall
- Iterative & Incremental Development
- Sample Processes
  + Unified Process
  + Spiral model

Software Product

- UML - History
- UML - Static
- UML - Dynamic

Literature

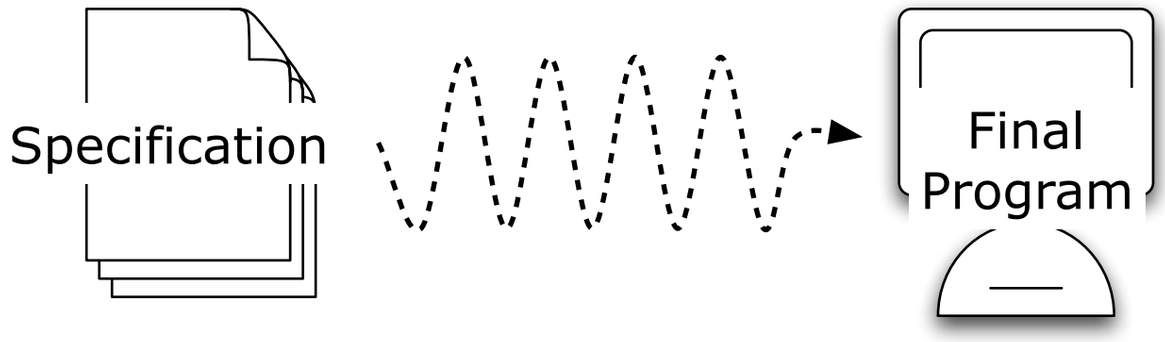
- [Ghez02a] — [Pres01a] — [Somm04a]
  Introductory chapters

Other

- [Brue00] Object-Oriented Software Engineering, B. Bruegge, A. Dutoit, Prentice Hall, 2000.
  - ➡ One of the first software engineering textbooks with a specific object-oriented perspective
- [Gold95a] Succeeding with Objects: Decision Frameworks for Project Management, A. Goldberg and K. Rubin, Addison-Wesley, 1995.
  - ➡ Explains how to define your own project management strategy

Why Software Engineering ?

A naive view on software development



But...

- Where did the specification come from?
- How do you know the specification corresponds to the user's needs?
- How did you decide how to structure your program?
- How do you know the program actually meets the specification?
- How do you know your program will always work correctly?
- What do you do if the users' needs change?
- How do you divide tasks if you have more than a one-person team?

What is Software Engineering ?

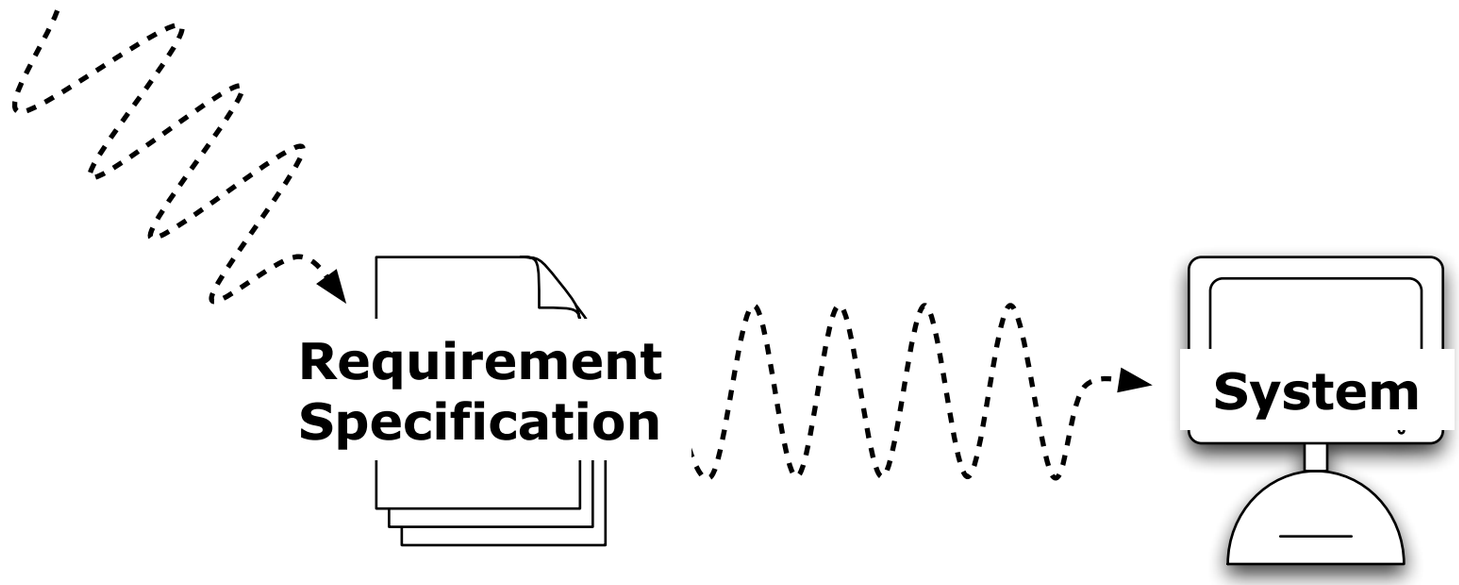
Some Definitions and Issues

- "state of the art of developing quality software on time and within budget" [Brue00]
  - Trade-off between perfection and physical constraints
    ➡ SE has to deal with real-world issues
  - State of the art!
    ➡ "best practice" is a moving target => life-long learning

- "multi-person construction of multi-version software" [Parnas]
  - Team-work
    ➡ Scale issue + Communication Issue
  - Successful software systems must evolve or perish
    ➡ Change is the norm, not the exception

- "software engineering is different from other engineering disciplines" [Somm04a]
  - Not constrained by physical laws
    ➡ limit = human mind
  - It is constrained by political forces
    ➡ balancing stake-holders

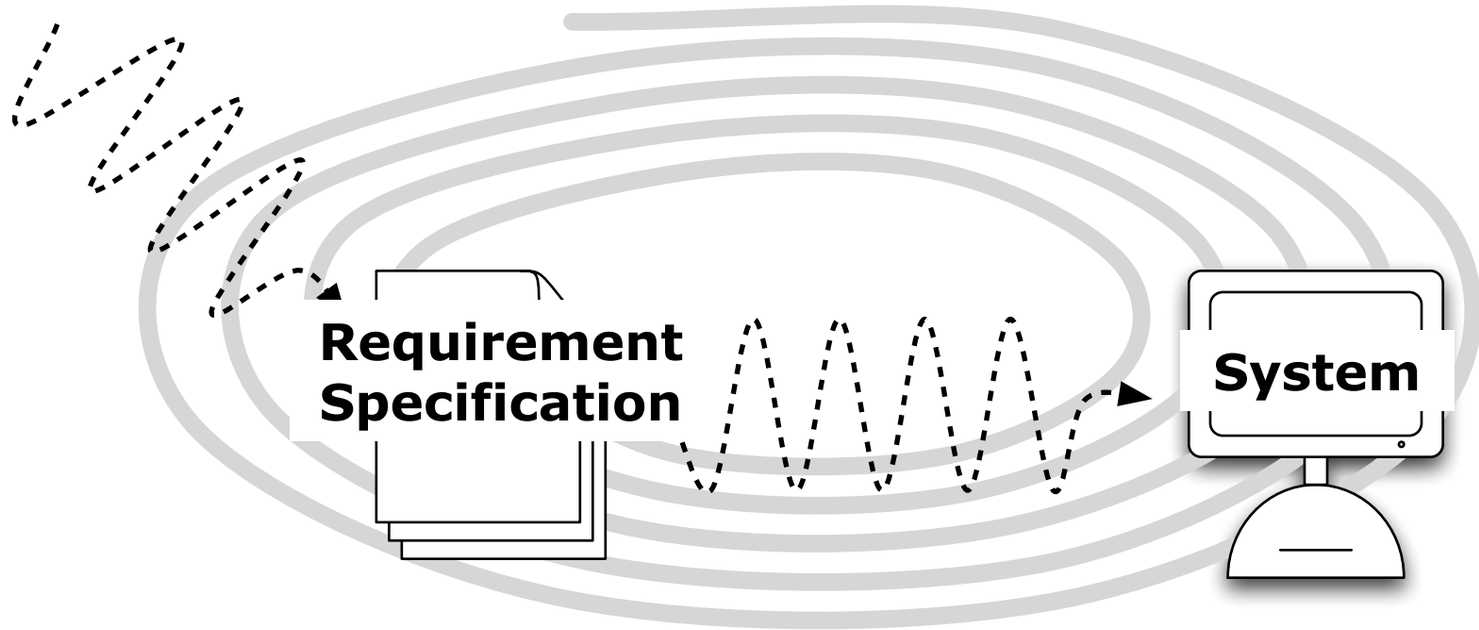
Product and Process



Product
= What is delivered to the customer
[Requirements Specification + System (+ all documentation, manuals, ...)]

Process
= Collection of activities that leads to (a part of) a product
[During process we apply techniques]

Evaluation Criteria



2 evaluation criteria to assess techniques applied during process

Correctness
- Are we building the right product?
- Are we building the product right?

Traceability
- Can we deduce which product components will be affected by changes?

Traceability

How to predict impact of changes ?

Maintain relationship

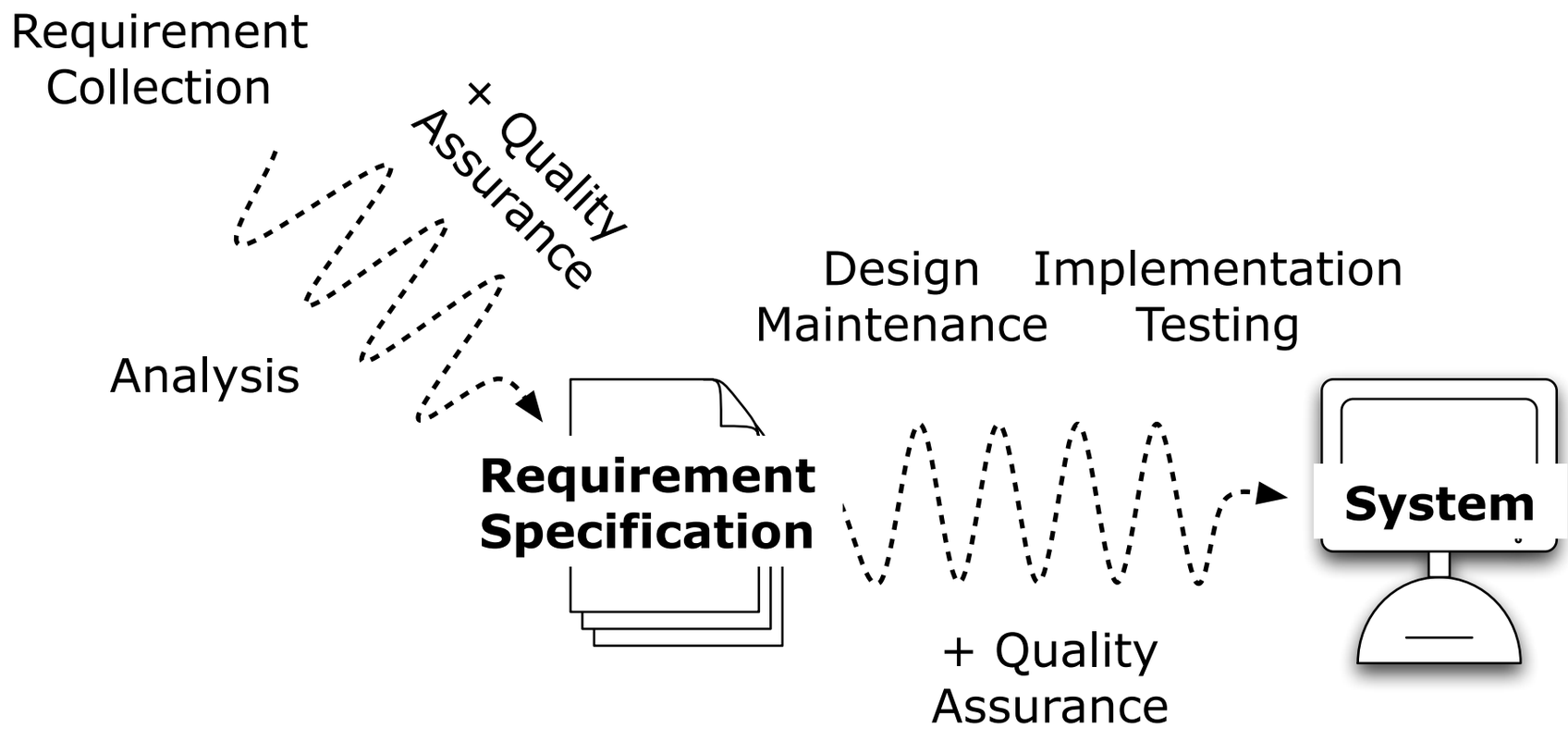
- from component to requirement that caused its presence
- from requirement that must be changed when component is adapted

	Comp 1	Comp 2	…	…	…	…	…	Comp m
Req 1				x				
Req 2	x							x
…								
…		x						
…						x	x	
Req n								x

This table is *virtual*: it is much too large to maintain explicitly !

⇒ A good process should help you deducing this relationship.

Software Process Activities (i)



Software Process Activities (ii)

Requirements Collection
- Establish customer's needs

Analysis
- Model and specify the requirements ("what")

Design
- Model and specify a solution ("how")
- system design (architecture) + detailed design (object design, formal spec)

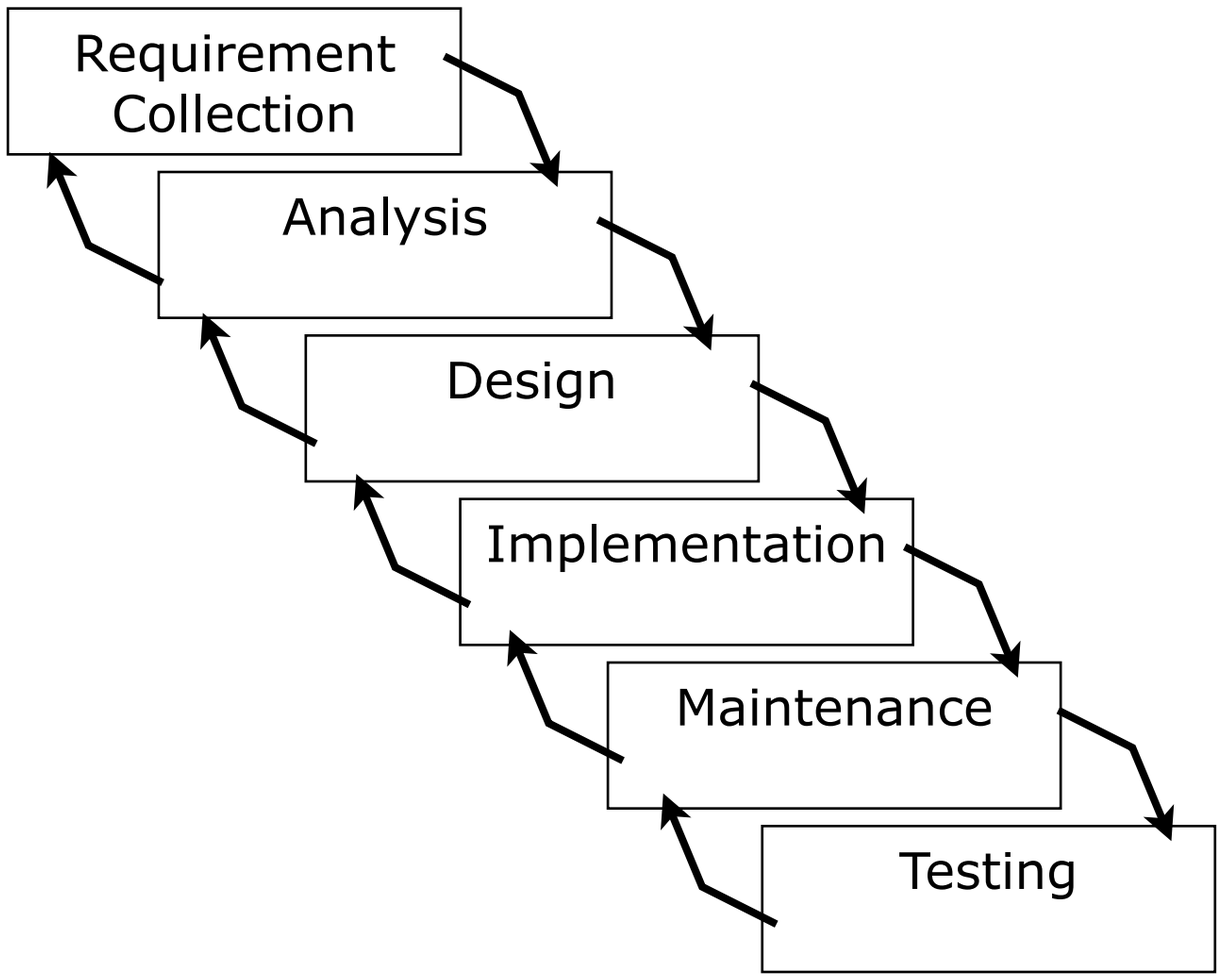
Implementation
- Construct a solution in software

Testing
- Verify the solution against the requirements

Maintenance
- Change a system after its been deployed
- = Repair defects + adapt to new requirements

Quality Assurance
- Make sure all above goes well
  = Deliver quality, on time and within budget

The Waterfall Software Lifecycle



The classical software life cycle models the software development as a step-by-step "waterfall" between the various development activities.

- going backward is possible but should be an exception (implies a mistake)

The waterfall model is popular for upper management, because

- *Visible*: it is easy to control project progress

The waterfall model is unrealistic for large projects, because

- *Complete*: a customer cannot state all requirements explicitly
- *Idealistic*: in real projects iteration occurs (but tools and organisation obstruct)
- *Time*: A working version of the system is only available late in the project
- *Change*: it is very difficult and costly to adapt to changes in the requirements

Iterative and Incremental Development

A good process must mix two principles (see [Gold95a], p. 94-96)

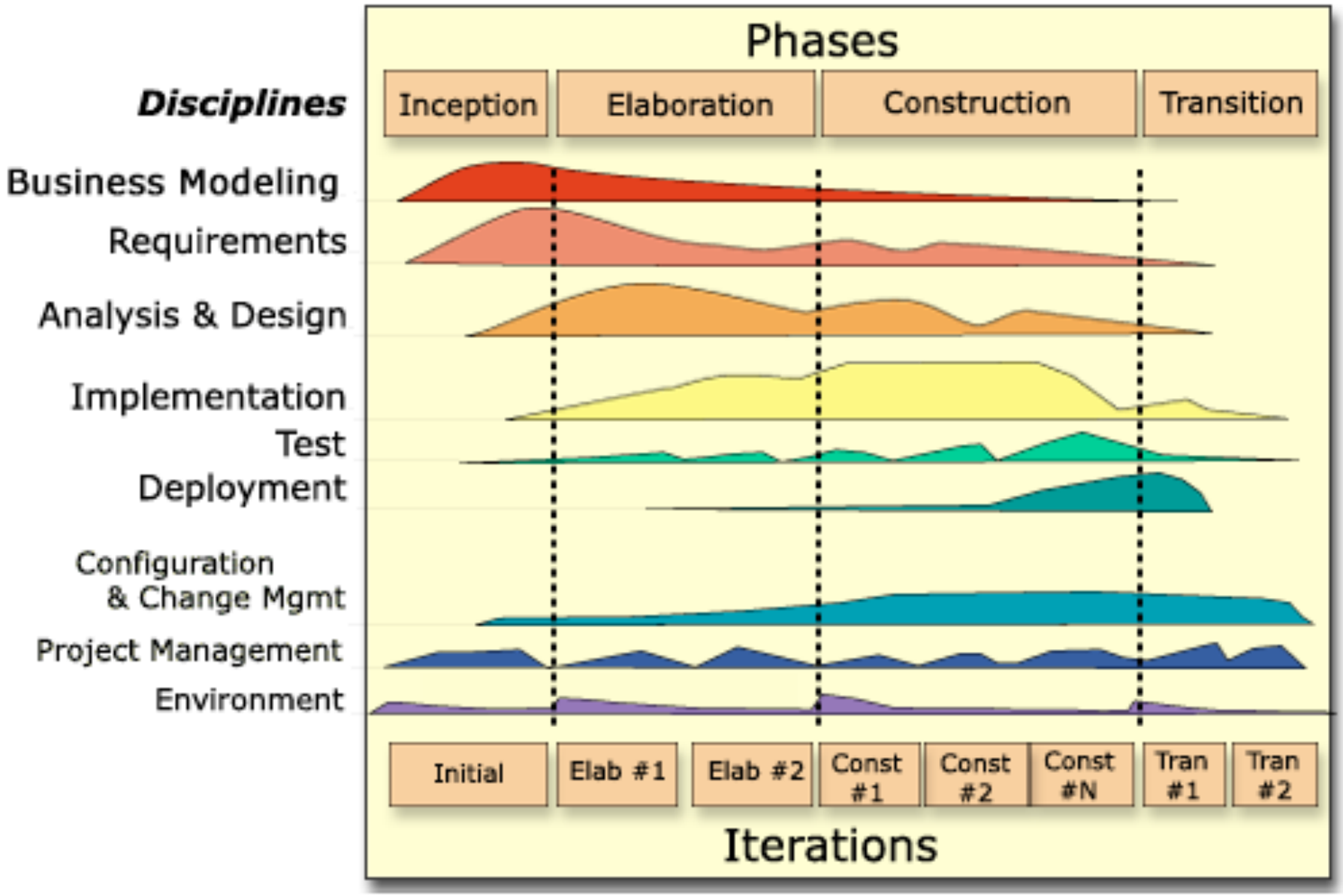
Iterative Development

- Controlled reworking of a system part to make improvements
  - ➡ We get things wrong before we get them right
    (Software development is a learning experience)

Incremental Development

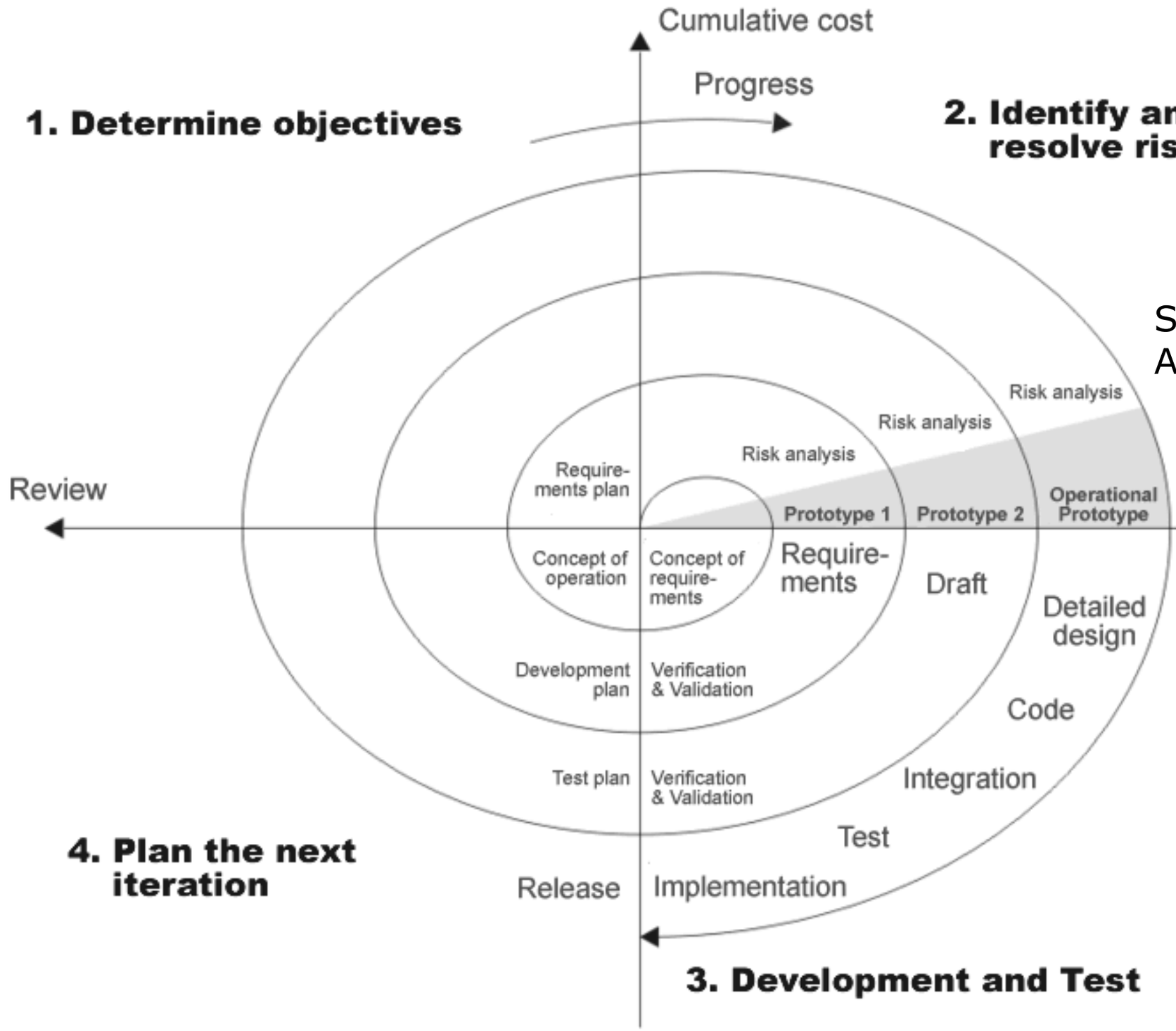
- Make progress in small steps to get early tangible results
  - ➡ Always have a running version
    (Control your learning via concrete intermediate steps)

The Unified Process



How do you plan the number of iterations? How do you decide on completion?

Boehm's Spiral Lifecycle



Stop ?
After risk analysis

go, no-go decision

Risk Analysis

Risk Identification

➡ Identify risk factors via “risk item checklist” (see [Pres01a])

- Project Risks: e.g., staffing risk
- Technical Risks: e.g. “leading edge” technology
- Business Risks: e.g., market risk (building a product that nobody wants)

Risk Projection (Risk Estimation)

➡ For each risk factor, estimate the probability and the impact

➡ Prioritize the list

- unimportant: low/moderate probability + low impact
- critical: high impact + moderate/high probability
- low impact + high probability

Risk Assessment

➡ For each critical risk factor, take action to reduce risk or terminate project

- Staff does not have the right skills -> Define training plan and hire extra staff
- “Leading edge” technology -> Build a prototype to evaluate benefits/drawbacks
- Market risk -> do a market study

Prototyping

A *prototype* is a software program developed to test, explore or validate a hypothesis, i.e. to reduce risks.

An *exploratory prototype*, also known as a throwaway prototype, is intended to validate requirements or explore design choices.

- UI prototype — validate user requirements
- rapid prototype — validate functional requirements
- experimental prototype — validate technical feasibility

An *evolutionary prototype* is intended to evolve in steps into a finished product

- grow, don't build [Broo87a]: "grow" the system redesigning and refactoring along the way
- combines incremental and iterative development

*** First do it, then do it right, then do it fast.

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.

http://agilemanifesto.org/

eXtreme Programming (XP)

Fine scale feedback

- Pair programming
- Planning game
- Test-driven development
- Whole team

Continuous process

- Continuous integration
- Refactoring or design improvement
- Small releases

Shared understanding

- Coding standards
- Collective code ownership
- Simple design
- System metaphor

Programmer welfare

- Sustainable pace

Coding

- The customer is always available
- Code the Unit test first
- Only one pair integrates code at a time
- Leave Optimization till last
- No Overtime

Testing

- All code must have Unit tests
- All code must pass all Unit tests before it can be released.
- When a Bug is found tests are created before the bug is addressed (a bug is not an error in logic, it is a test you forgot to write)
- Acceptance tests are run often and the results are published

SCRUM

Rugby metaphor

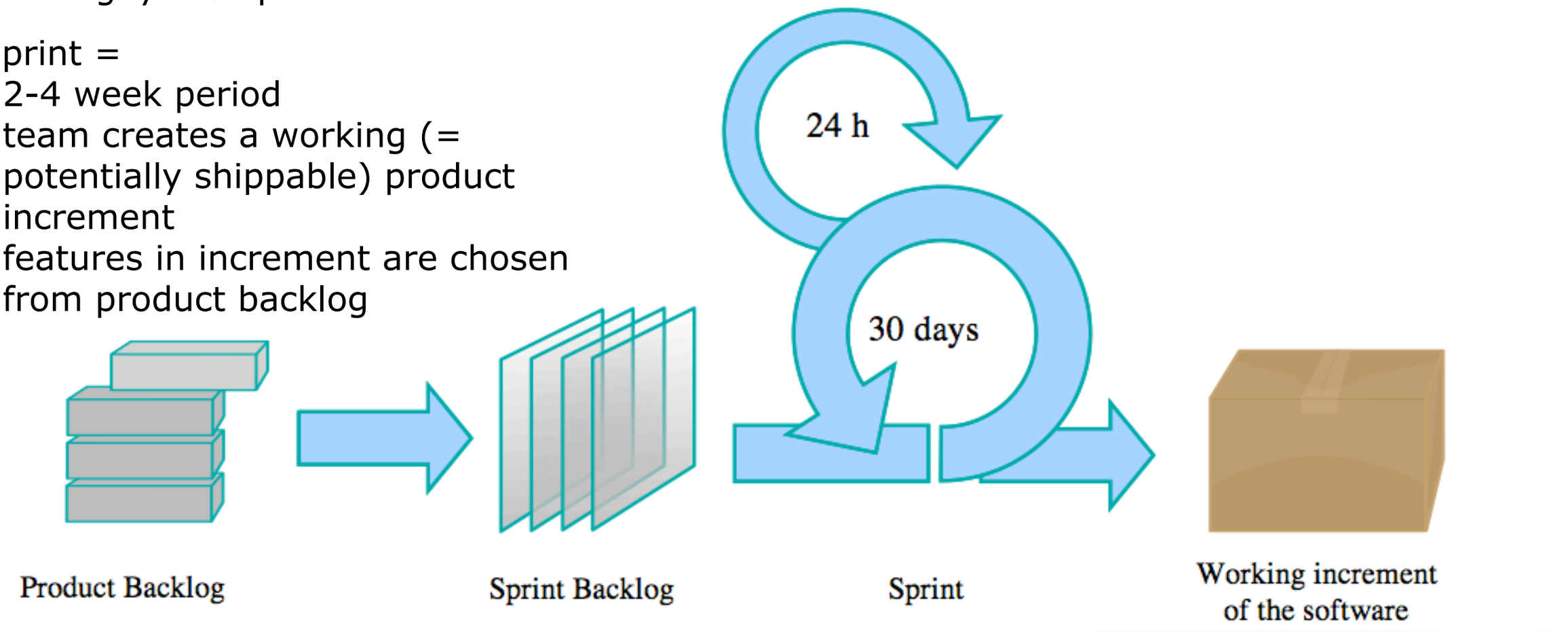
Sprint =

- 2-4 week period
- team creates a working (= potentially shippable) product increment
- features in increment are chosen from product backlog

SCRUM

Rugby metaphor

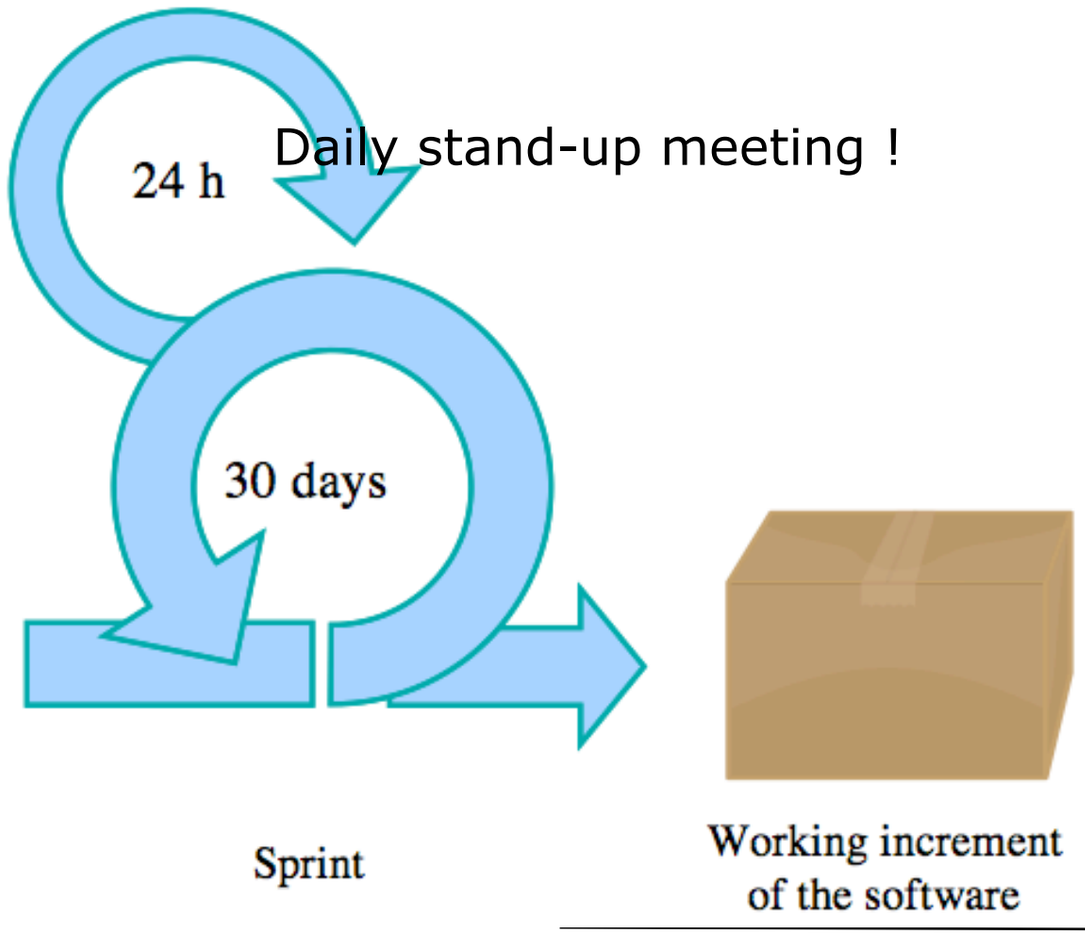
Sprint =
- 2-4 week period
- team creates a working (= potentially shippable) product increment
- features in increment are chosen from product backlog



SCRUM

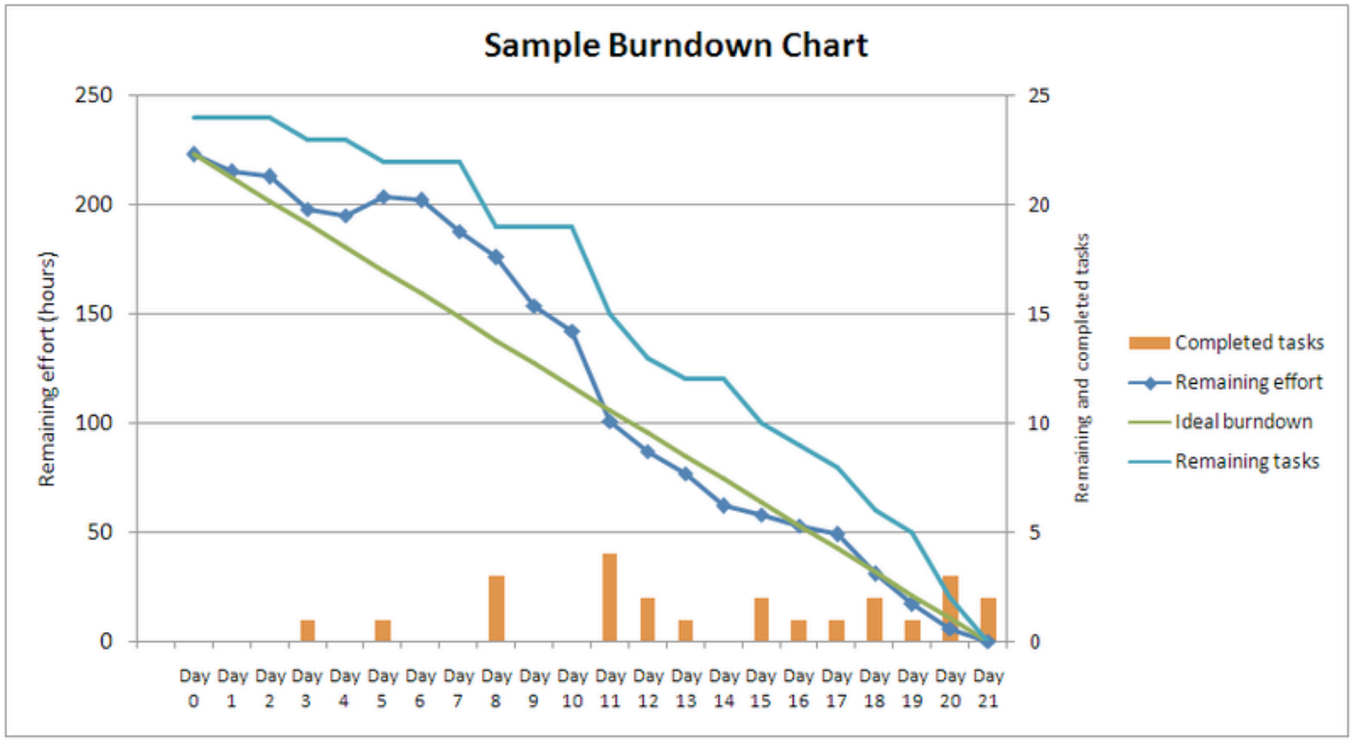
Rugby metaphor

Sprint =
- 2-4 week period
- team creates a working (= potentially shippable) product increment
- features in increment are chosen from product backlog



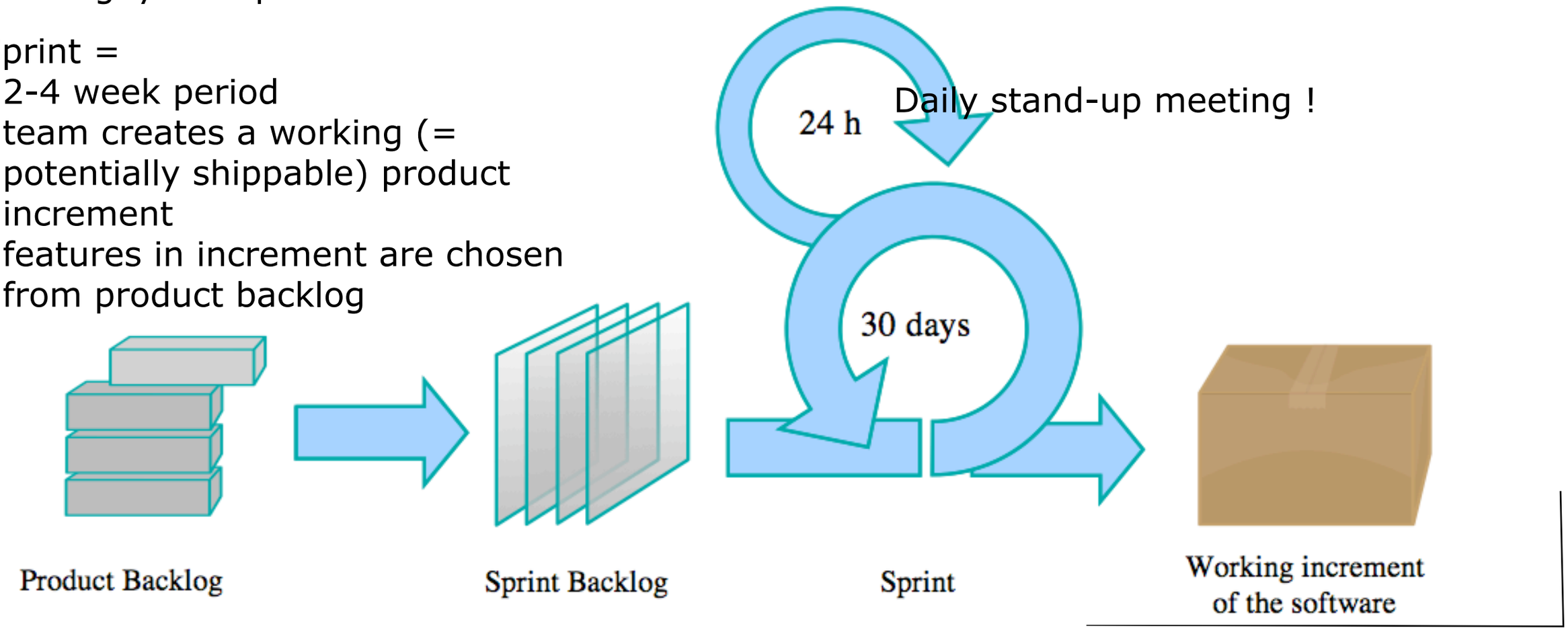
SCRUM

Rugby metaphor



Sprint =

- 2-4 week period
- team creates a working (= potentially shippable) product increment
- features in increment are chosen from product backlog



Agile or not ? There is no single truth …

Heavyweight

Lightweight

www.taxonweb.be

twitter

YouTube

Google

facebook

SOURCEFORGE

UML - History

First generation:

- Adaptation of existing notations (ER diagrams, state diagrams...):
  - ➡ Booch, OMT, Shlaer and Mellor,...
- Specialized techniques:
  - ➡ CRC cards; use-cases; design by contract

Second generation:

- Combination of "proven" ideas
  - ➡ Fusion: Booch + OMT + CRC + formal methods

Third generation:

- Unified Modeling Language:
  - ➡ uniform notation: Booch + OMT + Use Cases + Statecharts
  - ➡ complete lifecycle support (the Unified Process)
  - ➡ adaptable: you can extend the notation, choose your own process

Static UML - Classes (i)

Class name, attributes and operations:
(organized into compartments)

Polygon
centre: Point vertices: List of Point borderColour: Colour fillColour: Colour
display (on: Surface) rotate (angle: Integer) erase ( ) destroy ( ) select (p: Point): Boolean

A collapsed class view.
(NB: attributes & operations not shown, so don't know whether empty or not!)

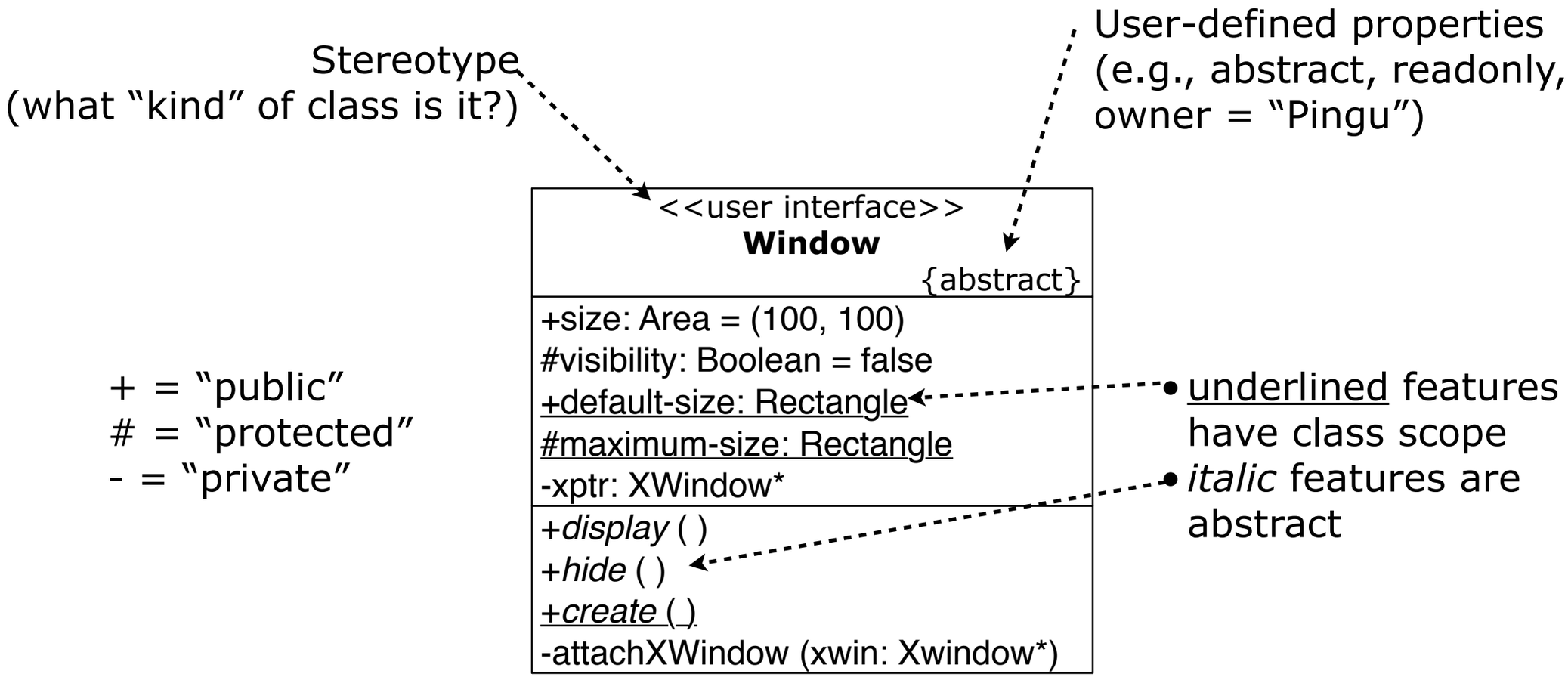
Polygon

Class with Package name:
(Optional, but useful for large systems !)

ZWindows::Window

Attributes and operations are also collectively called features.

Static UML - Classes (ii)

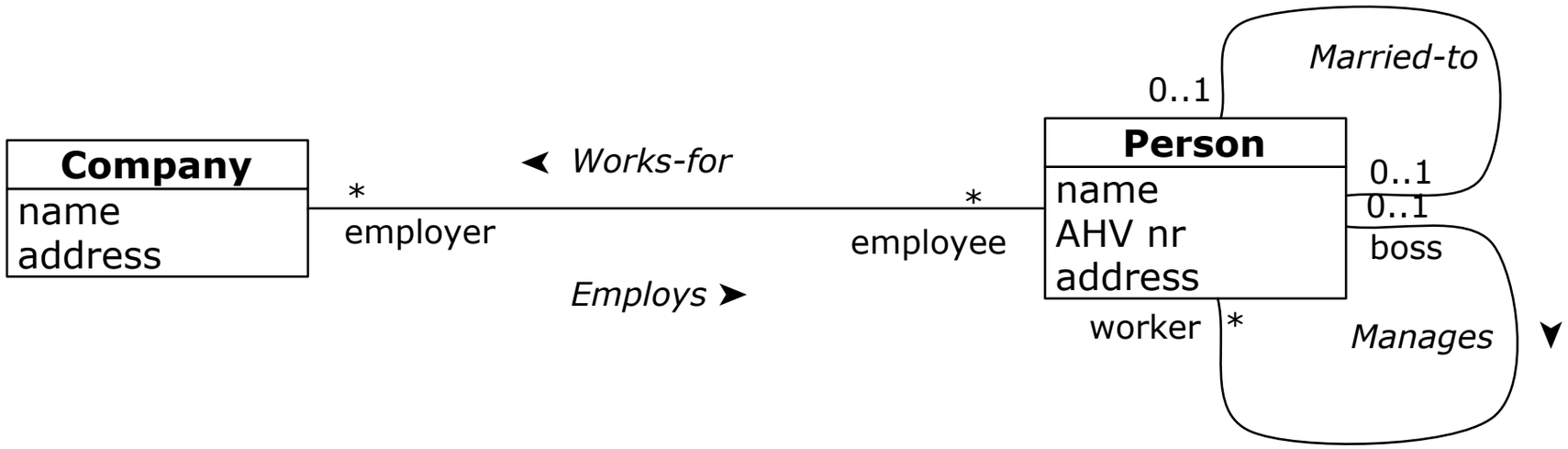


- Attributes are specified as: name: type = initialValue { property string }
- Operations are specified as: name (param: type = defaultValue, ...) : resultType

Static UML - Associations

Associations

- denoted by a solid line.
- represents structural relationships between objects of different classes.



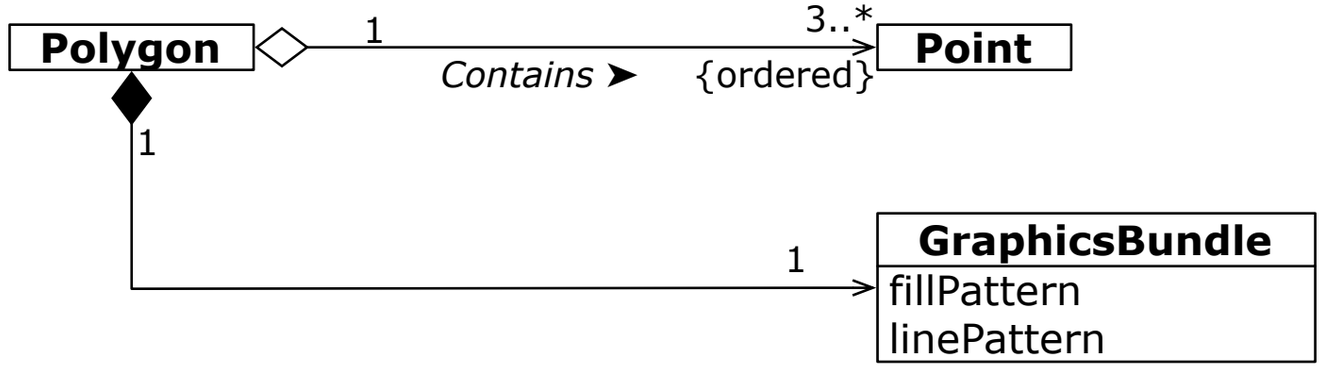
- optional *name* and *direction*
- (unique) *role names* and *multiplicities* at end-points (BEWARE POSITION)
- traverse using *navigation expressions*

  e.g., universityAntwerp.employee[name = “Demeyer”].wife

Static UML – Aggregation & Composition

Aggregation
- denoted by a hollow diamond
- whole-part relationship: part may exist without the whole (i.e. whole owns a reference to the part)

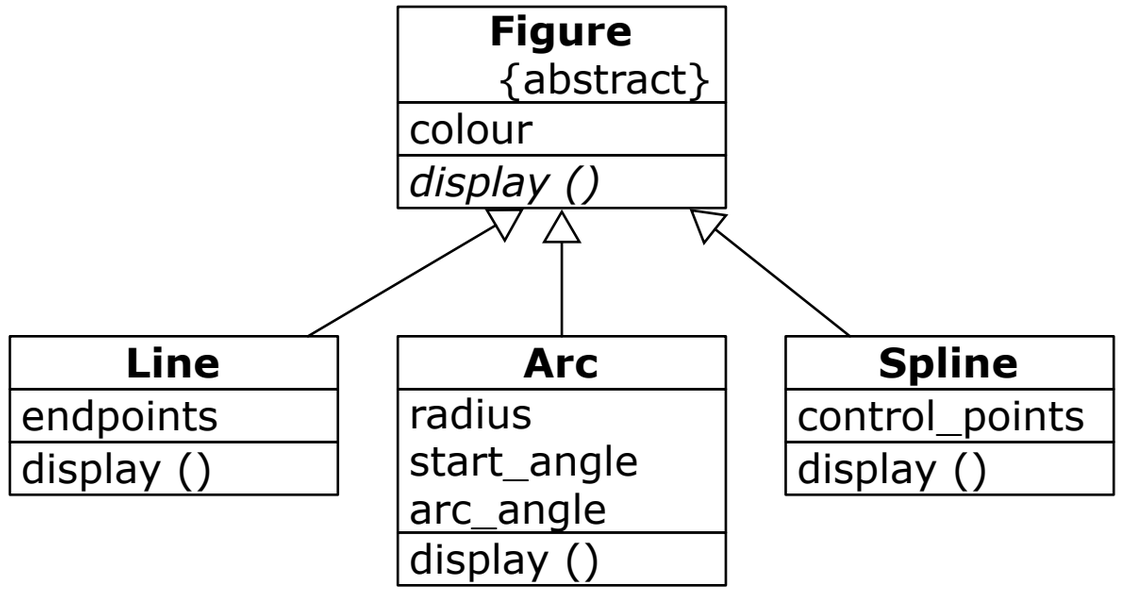
Composition
- denoted by a solid diamond
- whole-part relationship: part must always exist with the whole (i.e., whole owns the part)



Static UML - Generalization

Generalization

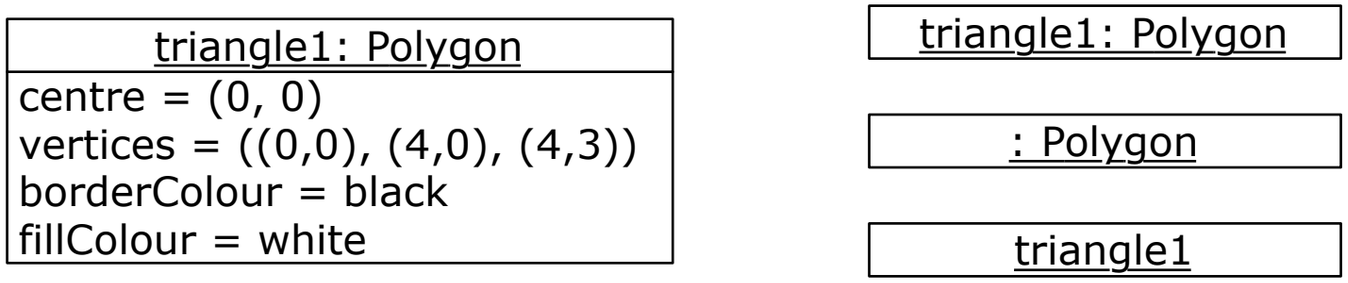
- denoted with a hollow arrow from the specific to the general
- represents inheritance, is-a relationships, code reuse relationship (philosophical debate: Square inherits from Rectangle or vice-versa)



Dynamic UML - Objects

Objects

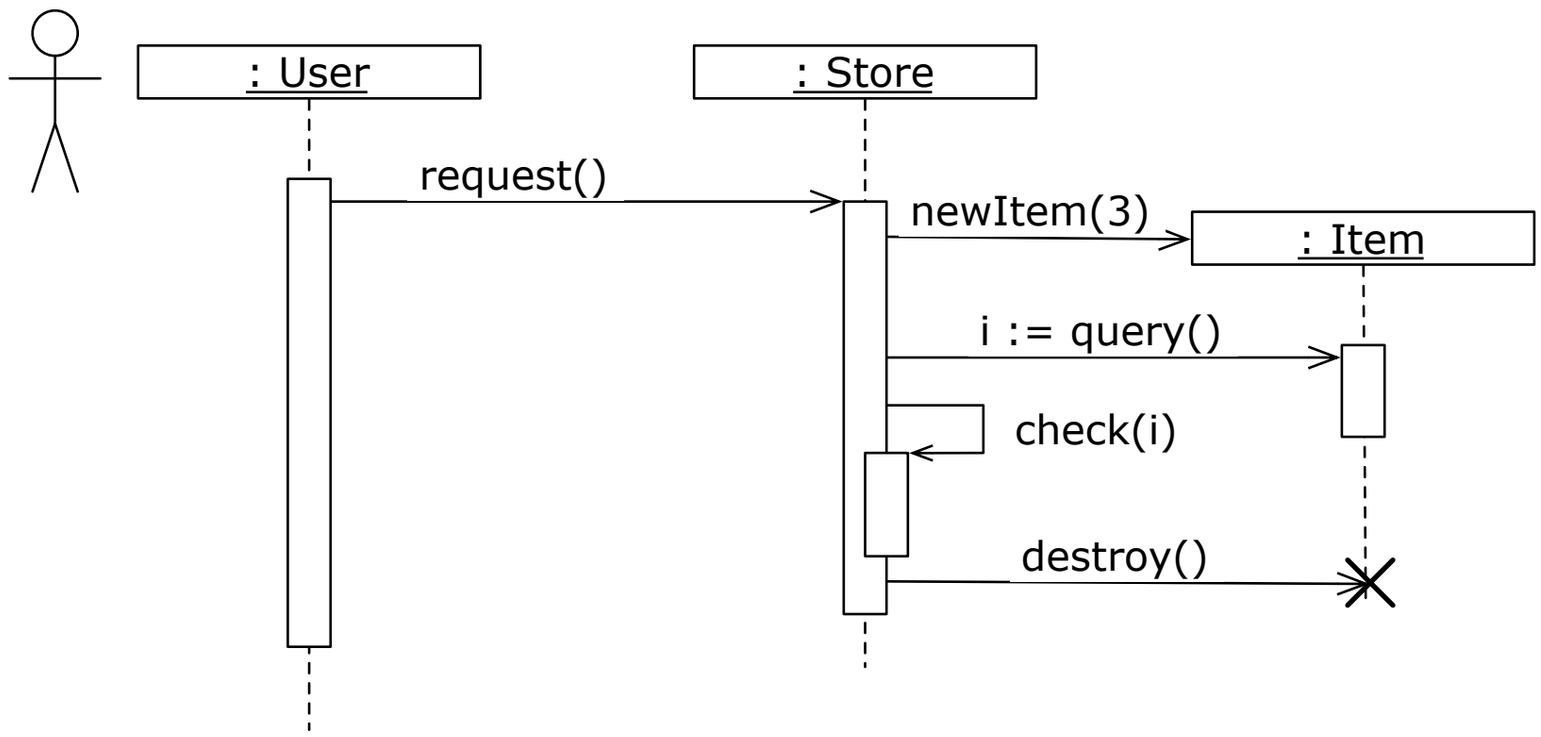
- shown as rectangles with their name and type underlined in one compartment
- attribute values, optionally, in a second compartment
- the name of the object may be omitted (then colon must be kept with class name)
- the class of the object may be supressed (together with the colon) to represent an anonymous object

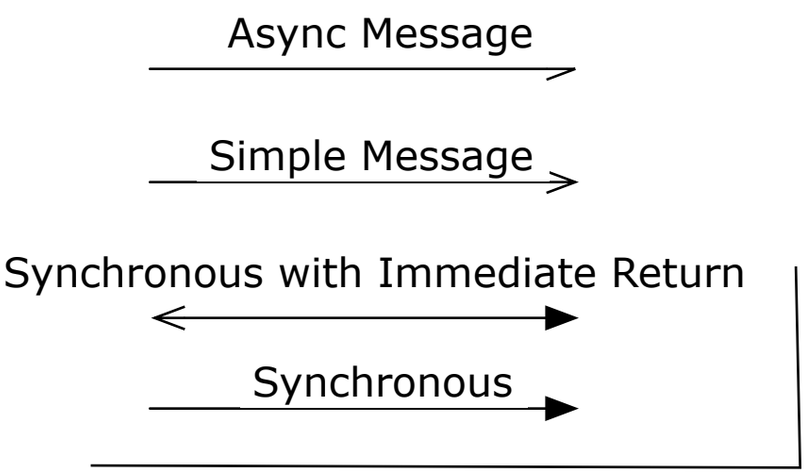


Dynamic UML - Sequence Diagrams

Sequence Diagrams

- Object at top, lifeline as dashed vertical line (time flows from top to bottom)
- Method execution as rectangle, message sends as arrow with message name
- Possibility to show concurrency via special arrowheads



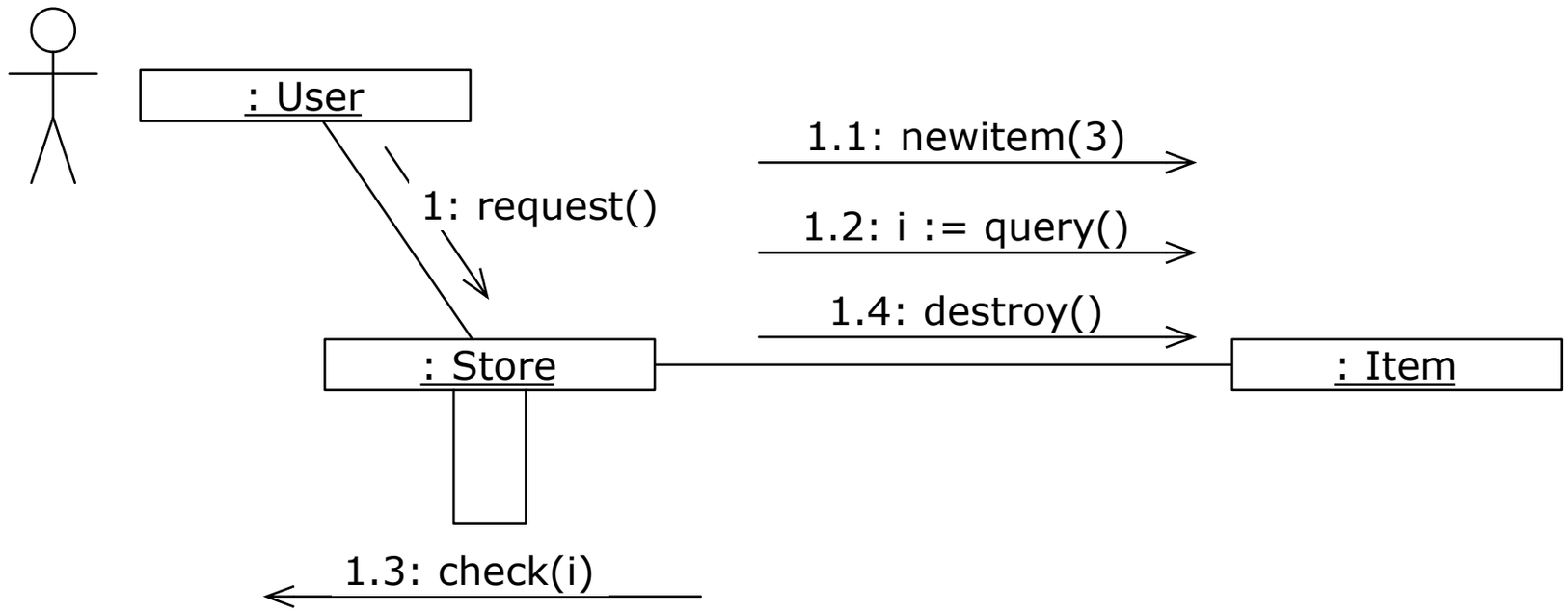


Dynamic UML - Collaboration Diagrams

Collaboration Diagrams

- Objects with associations positioned freely in the diagram
- Messages with little arrows near to associations
- Message sequences follow from hierarchical numbering
- Expressibility is identical to sequence diagrams

⇒Freedom in lay-out but message sequence difficult to follow



Summary (i)

You should know the answers to these questions:

- How does Software Engineering differ from programming?
- Why is programming only a small part of the cost of a "real" software project ?
- Give a definition for "traceability".
- What is the difference between analysis and design?
- Why is the "waterfall" model unrealistic? Why is it still used?
- What's the relationship between iterative development, incremental development and (evolutionary) prototyping?
- How do you decide to stop in the spiral model?
- How do you identify risk ? What do you do when projecting a risk? Which risks do you assess as critical?
- List the 6 principles of extreme programming.
- What is a "sprint" in the SCRUM process ?
- Draw a UML class diagram modeling marriages in cultures with monogamy (1 wife marries 1 husband), polygamy (persons can be married with more than one other person), polyandry (1 woman can be married to more than one man) and polygyny (1 man can be married to more than one woman).
- Draw a UML diagram that represents an object "o" which creates an account (balance initially zero), deposits 100$ and then checks whether the balance is correct.

Summary (ii)

Can you answer the following questions?

- What is your preferred definition of Software Engineering? Why?
- Why do we choose “Correctness” & “Traceability” as evaluation criteria? Can you imagine some others?
- Why is “Maintenance” a strange word for what is done during the activity?
- Why is risk analysis necessary during incremental development?
- How can you validate that an analysis model captures users’ real needs?
- When does analysis stop and design start?
- When can implementation start?
- Can you compare the Unified Process and the Spiral Model ?
- Can you explain the values behind the Agile Manifesto ?
- Can you identify some synergies between the techniques used during extreme programming ?
- Is it possible to apply Agile Principles with the Unified Process ?
- Did the UML succeed in becoming *the* Universal Modeling Language ? Motivate your answer.