MSO
Analysis & UML

Hans Philippi
(based on the course slides of Wouter Swierstra)

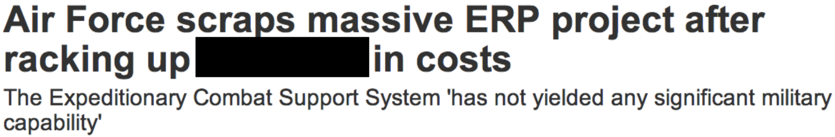
August 24, 2018

Recap: Last lectures

- How can I manage the process of constructing complex software?
- Waterfall model, the Unified Process, iterative development, embracing change
- How do I know what the customer wants?
- Requirement analysis, use cases, scenarios

Let us start with some motivating examples for the next topics . . .

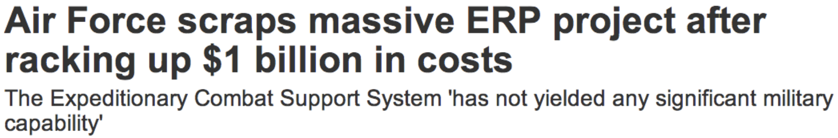
Fill in the blanks...


Air Force scraps massive ERP project after racking up [redacted] in costs

The Expeditionary Combat Support System 'has not yielded any significant military capability'

ERP stands for Enterprise Resource Planning – think of software to manage accounting, HR, etc.
(Source `http://www.computerworld.com.au`)

Disaster tourism



Air Force scraps massive ERP project after racking up $1 billion in costs

The Expeditionary Combat Support System 'has not yielded any significant military capability'

Disaster tourism

> *We estimate it would require an additional $1.1B for about a quarter of the original scope to continue and fielding would not be until 2020. The Air Force has concluded the ECSS program is no longer a viable option... Therefore, we are cancelling the program and moving forward with other options...*

An Air Force spokesman

Disaster tourism

The system dates back to 2005, when Oracle won an $88.5 million software contract, securing the deal over rival SAP and other vendors. It was supposed to replace more than 200 legacy systems.

Disaster tourism

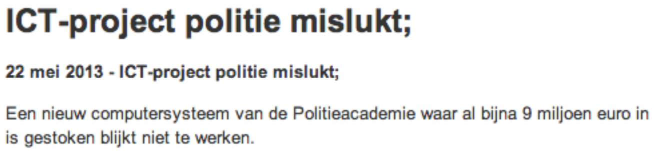
> *"This situation raises more questions than answers," Krigsman said. "Why did it take the [Air Force] $1 billion and almost 10 years to realize this project is a disaster? What kind of planning process accepts a billion dollars of waste?"*

Michael Krigsman, expert on IT project failures

Disaster tourism

Of course, this is only a problem for really large organisations, like the US Airforce, right?

Disaster tourism

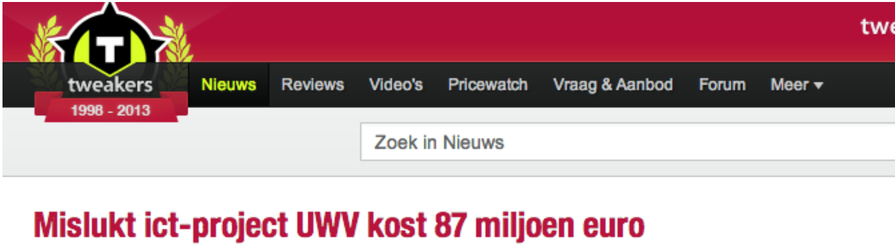


ICT-project politie mislukt;

22 mei 2013 - ICT-project politie mislukt;

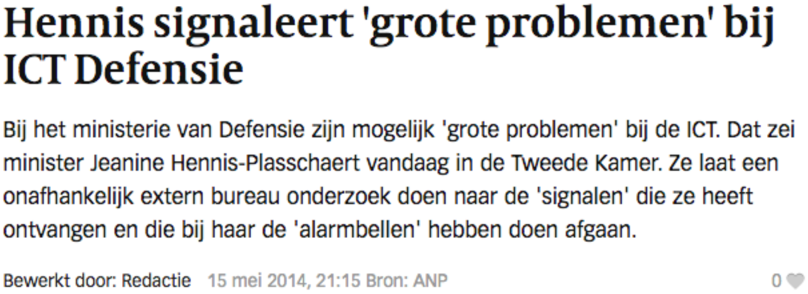
Een nieuw computersysteem van de Politieacademie waar al bijna 9 miljoen euro in is gestoken blijkt niet te werken.

Disaster tourism



Mislukt ict-project UWV kost 87 miljoen euro

Disaster tourism



Hennis signaleert 'grote problemen' bij ICT Defensie

Bij het ministerie van Defensie zijn mogelijk 'grote problemen' bij de ICT. Dat zei minister Jeanine Hennis-Plasschaert vandaag in de Tweede Kamer. Ze laat een onafhankelijk extern bureau onderzoek doen naar de 'signalen' die ze heeft ontvangen en die bij haar de 'alarmbellen' hebben doen afgaan.

Bewerkt door: Redactie 15 mei 2014, 21:15 Bron: ANP 0

Disaster tourism

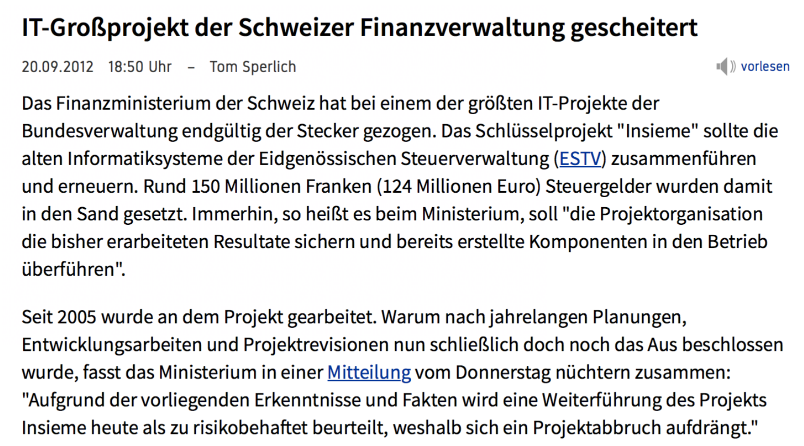
ICT-project basisregistratie totaal mislukt

Modernisering bevolkingsregister

Minister Plasterk stopt groot project voor modernisering van het bevolkingsregister, na vele waarschuwingen en advies van een commissie. 90 miljoen lijkt weggegooid.

Liza van Lonkhuyzen 7 juli 2017

Disaster tourism



IT-Großprojekt der Schweizer Finanzverwaltung gescheitert

20.09.2012 18:50 Uhr – Tom Sperlich

vorlesen

Das Finanzministerium der Schweiz hat bei einem der größten IT-Projekte der Bundesverwaltung endgültig der Stecker gezogen. Das Schlüsselprojekt "Insieme" sollte die alten Informatiksysteme der Eidgenössischen Steuerverwaltung (ESTV) zusammenführen und erneuern. Rund 150 Millionen Franken (124 Millionen Euro) Steuergelder wurden damit in den Sand gesetzt. Immerhin, so heißt es beim Ministerium, soll "die Projektorganisation die bisher erarbeiteten Resultate sichern und bereits erstellte Komponenten in den Betrieb überführen".

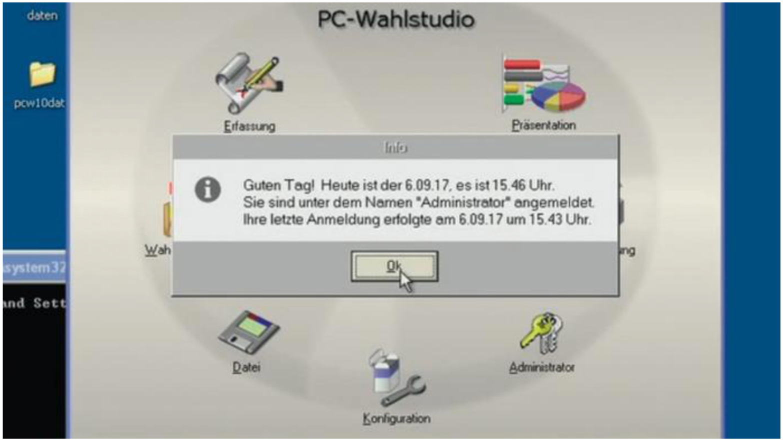
Seit 2005 wurde an dem Projekt gearbeitet. Warum nach jahrelangen Planungen, Entwicklungsarbeiten und Projektrevisionen nun schließlich doch noch das Aus beschlossen wurde, fasst das Ministerium in einer Mitteilung vom Donnerstag nüchtern zusammen: "Aufgrund der vorliegenden Erkenntnisse und Fakten wird eine Weiterführung des Projekts Insieme heute als zu risikobehaftet beurteilt, weshalb sich ein Projektabbruch aufdrängt."

Disaster tourism

Mängel in der Wahlsoftware seit Monaten bekannt

16.09.2017 15:40 Uhr – Ulrich Hilgefort

vorlesen



Guten Tag! Heute ist der 6.09.17, es ist 15.46 Uhr.
Sie sind unter dem Namen "Administrator" angemeldet.
Ihre letzte Anmeldung erfolgte am 6.09.17 um 15.43 Uhr.

PC-Wahl-Software

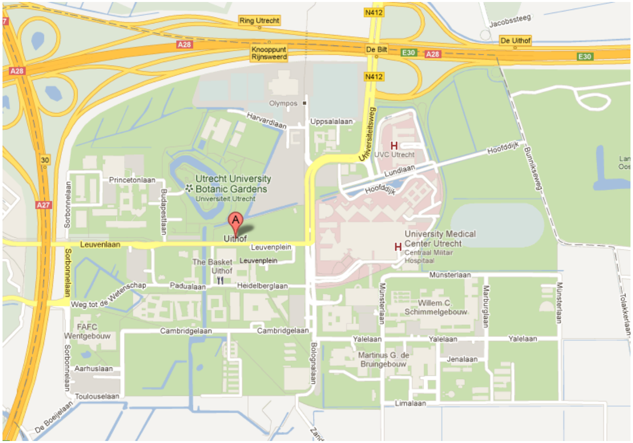
This lecture

- How can we use our requirements and use cases to *analyse* and *design* software?
- What *notation* can we use to describe complex systems concisely?

Analysis

- The aim of analysis is to come up with an abstract *model* of the problem domain
- You want to try and transfer requirements and use cases (written with the customer) to a technical specification (written for developers)
- A good model *simplifies* reality, but represents all relevant data

Model: example



Why model software?

- Software is too complex to oversee all the details at once: we need abstractions!
  - Windows 8 has about 50 million lines of code
  - Linux kernel has about 15 million lines of code
- Code is written for a machine to execute – not necessarily for a human to understand

Modeling software?

But how can we model software?

The Unified Modeling Language

- The UML is a single language with 14(!) different ‘dialects’:
  - use case diagrams
  - activity diagrams
  - class diagrams
  - sequence diagrams
  - and several others
- It is the *de facto* standard way for designers, developers, business analysts, and any other technical stakeholders to communicate

UML: Unified Modeling Language

Before describing the process of analysis further, we need to describe some more the UML notation

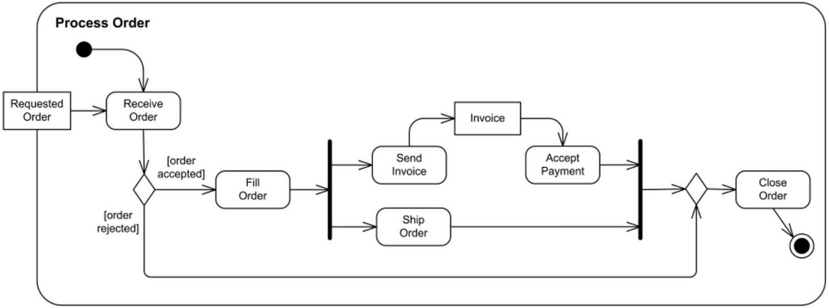
The UML – Activity diagrams

Activity diagrams are useful to describe existing business processes. This can help to validate that you and your customer agree on how the current business works

The UML – Activity diagrams – Basic elements

●	A single black dot - the initial state
◉	A black dot with a white circle - the final state
state	Rounded rectangles - activities ("file form", "ship order", etc.)
DataObject	Rectangles - objects that are part of the workflow (form, invoice, ticket, etc.)
◇	Diamonds - choice (with labelled outgoing edges) or merge points
▬	Black bars (horizontal or vertical) - split and join of concurrent activities
→	Directed edges (labelled and unlabelled) to connect nodes

The UML – Activity diagrams – example



UML – Activity diagrams

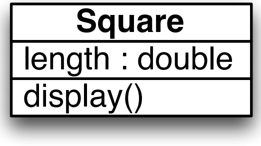
Activity diagrams are particularly useful during the *Business Modeling* phase of the RUP – where you establish how the current business processes work

The UML is not *only* about software development!

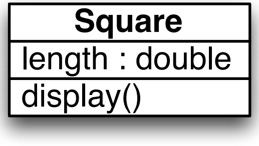
UML – Class diagrams

- One of the most common UML diagrams is the *class* diagram, used to give a high-level overview of the software system
- A class diagram consists of:
  - boxes to represent classes (together with their attributes and methods)
  - various arrows to describe the relation between classes
- These diagrams are drawn at a varying level of detail, depending on the situation

UML – Class diagrams

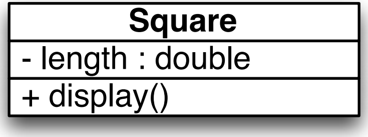


UML – Class diagrams



```
public class Square
{
    public double length;
    public void display()
        {
            \\ code for display
        }
}
```

UML – Class diagrams

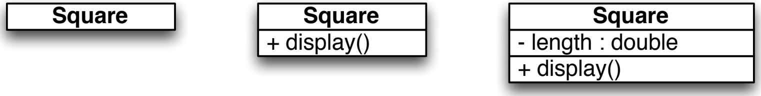


Use modifiers to indicate the visibility of methods and attributes:

- + indicates a method or attribute is public
- − indicates a method or attribute is private
- # indicates a method or attribute is protected

There are further modifiers for packages, static methods, etc
Further details can be found in the UML documentation linked from the website

Different levels of detail

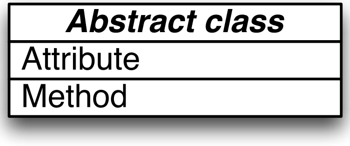


Choose the right level of detail, for the right situation:

- Early sketches: just use class name
- Full design: complete overview of attributes and methods

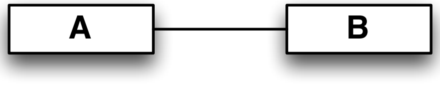
Goal: communicate unambiguously with developers

Abstract classes



Print the class name in an *italic* to define that class abstract

Relations between objects

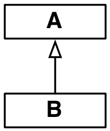


The objects A and B are connected ‘somehow.’

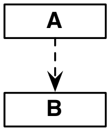
But can we be more specific?

Specific relations between objects

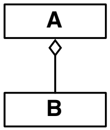
Inheritance



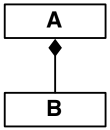
Dependency



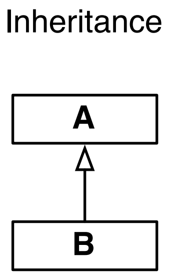
Aggregation



Composition



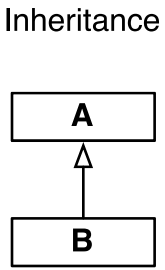
Inheritance



Pronounced:

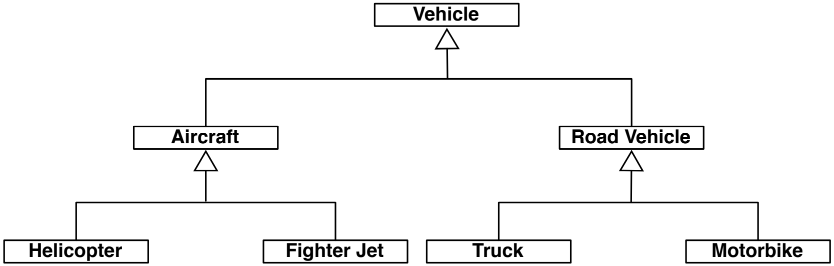
- B is a subclass of A
- B is derived from A
- A is the superclass of B

Inheritance – example



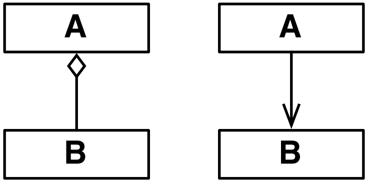
```
public class B : A
{
    // Class implementation goes here
}
```

Inheritance – example



Aggregation and Association

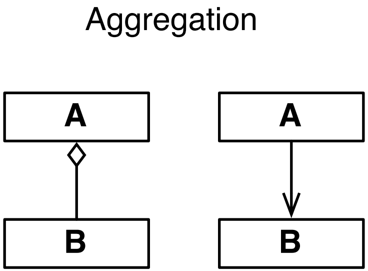
Aggregation



Read:

- A has a B
- B is a part of A

Aggregation and Association



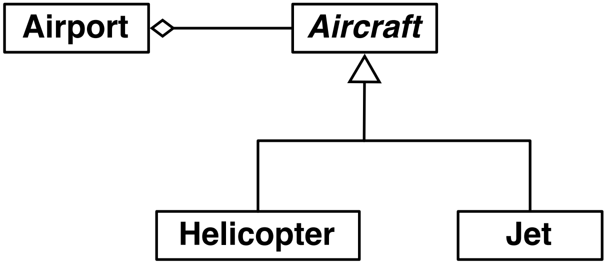
```
public class A
{
    public B myB;
}
```

Aggregation or attribute?

Typically you should use:

- *attributes* to describe primitive types
- *aggregation* to describe the relation with other classes

Aggregation – example



Note that:

- The Aircraft class is kept abstract;
- Jets and Helicopters are subclasses of Aircraft.

A picture is worth a thousand words

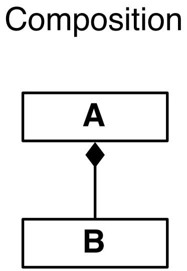
UML this

```
abstract class Shape {
  private int width;
  private int height;
  public Color c;
  public int area()
}

public class Circle : Shape
...

public class Rectangle : Shape
...
```

Composition



Composition is a special form of aggegration, where the contained object is a part of the containing object (and has no right to exist by itself or be referenced by other objects).
The distinction is especially important in languages without garbage collection – who is responsible for the clean-up?

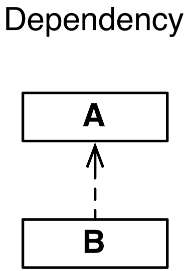
Association vs Composition

Sketch the classes and its relations for the following cases.

We own a fleet of planes. On of the plane types is the Boeing 737. Each 737 has a left engine and a right engine. Each plane and each engine have known identities (and a maintenance record).

Our catalog has a collection of racing bikes (for instance X-LITE CRS-3000). Each racing bike has a rear derailleur (for instance Ultegra 6800).

Dependency



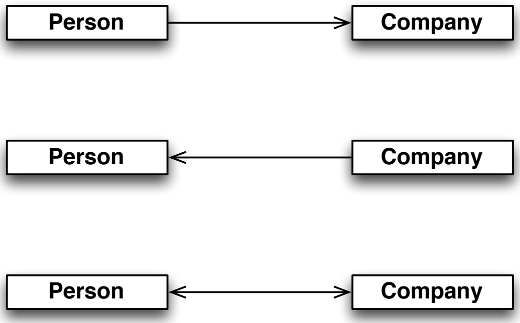
Read:

- B depends on A
- B uses A

Dependency compared to aggregation

- The definition of *aggregation* is unambiguous, *dependency* is more subtle
- Typically dependency is used when there is some relation between two objects, but not this is not an aggregation relation, for example:
    - Objects of class B create objects of type A
    - Methods in class B take arguments of type A
    - Or the class B calls static methods of the class A
- More generally, if a change to class A *could* affect class *B*, then B depends on A
- A good design minimizes dependencies

Directed associations

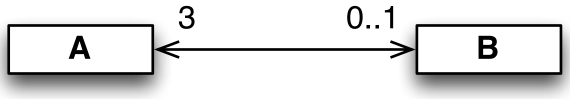


These pictures correspond to the following three situations:

- A person knows the company for which he works
- A company knows its employees
- Both person and company know about each other

Cardinalities

You may associate a number or range with any association, e.g.

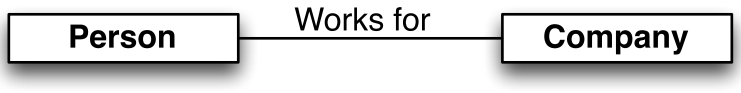


Every A has 0 or 1 objects of type B
Every B has 3 objects of type A
Other examples:

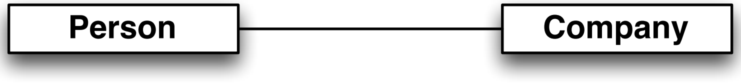
- A car has four wheels
- You may borrow at most six books from the library

Labelled associations

Sometimes you may want to label an association:



Undirected associations



A relation without arrow is said to be *undirected*.
This can mean two things:

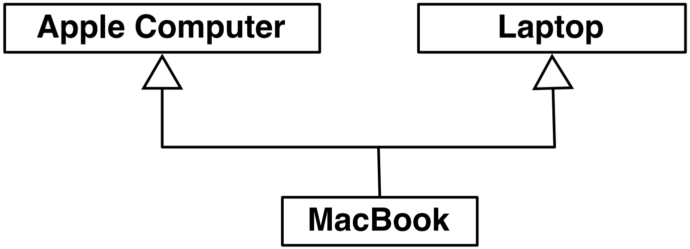
- The exact relation is undecided
- Both objects know about one another

We will work with the first definition.
In the final design, avoid any such ambiguity!

The UML vs code

- The UML is a great way to sketch out ideas. You can start very abstractly (just boxes and lines), and flesh out the design as you go along
- A fully worked out UML class diagram can be translated to code (with empty method bodies)
- Be careful – it is easy to write UML diagrams that cannot be implemented

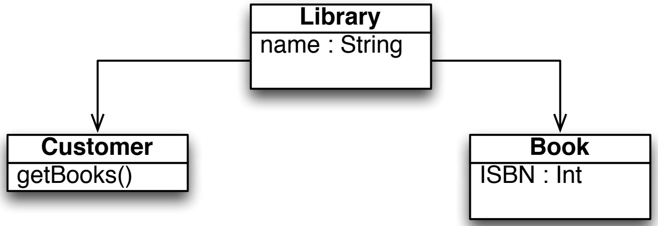
Problem?



Problem!

A class may only have one superclass.

Problem?



Make sure every method has the information it needs

A class can only access:

- Its own private, public, or protected attributes/methods
- The public attributes/methods of other classes it has (aggregation)
- The protected and public attributes/methods of its superclasses

Writing precise designs

- A design written using the UML is more than just a pretty picture – it needs to be implemented using code
- Compiler may provide some sanity check that a program will execute. When you write a design, there is no compiler to check that your design makes sense.
- Convince yourself:
  - check for common errors
  - replay use cases in your design
  - write explicit *sequence diagrams* and check that they are in accordance with the design

Epilog

- We have met with UML class diagrams to model software
- Next lecture: we will see other variants of UML diagrams