



## Lecture 12

# PROCESSES AND ADVANCED TECHNIQUES

PB007 Software Engineering I  
Faculty of Informatics, Masaryk University  
Fall 2015

# Topics covered

---



- ✧ Summary of covered topics
- ✧ Software process models
- ✧ Outline of advanced techniques
- ✧ Tool support
- ✧ Course follow-up



---

# Summary of Covered Topics

## Lecture 12/Part 1

# Covered topics



1. **Software development**, UML Use Case diagram.
2. **Requirements specification**, UML Activity diagram.
3. System analysis and design, structured vs. object-oriented A&D.
4. **Object oriented analysis**, UML Class, Object and State diagram.
5. **Structured analysis**, data modelling, ERD.
6. **High-level design**, UML Class diagram in design.
7. **Low-level design and implementation**, UML Interaction diagrams
8. **Architecture design**, UML Package, Component and Deployment diagram.
9. **Testing**, verification and validation.
10. **Operation**, maintenance and system evolution.



Software development **management**.



# Software Process Models

## Lecture 12/Part 2

# Software process models



## ✧ The waterfall model

- Plan-driven model. Separate and distinct phases of specification and development.

## ✧ Incremental development

- Specification, development and validation are interleaved. May be plan-driven or agile.

## ✧ Reuse-oriented software engineering

- The system is assembled from existing components. May be plan-driven or agile.

✧ In practice, most large systems are developed using a process that incorporates elements **from many different models.**

# Plan-driven and agile development



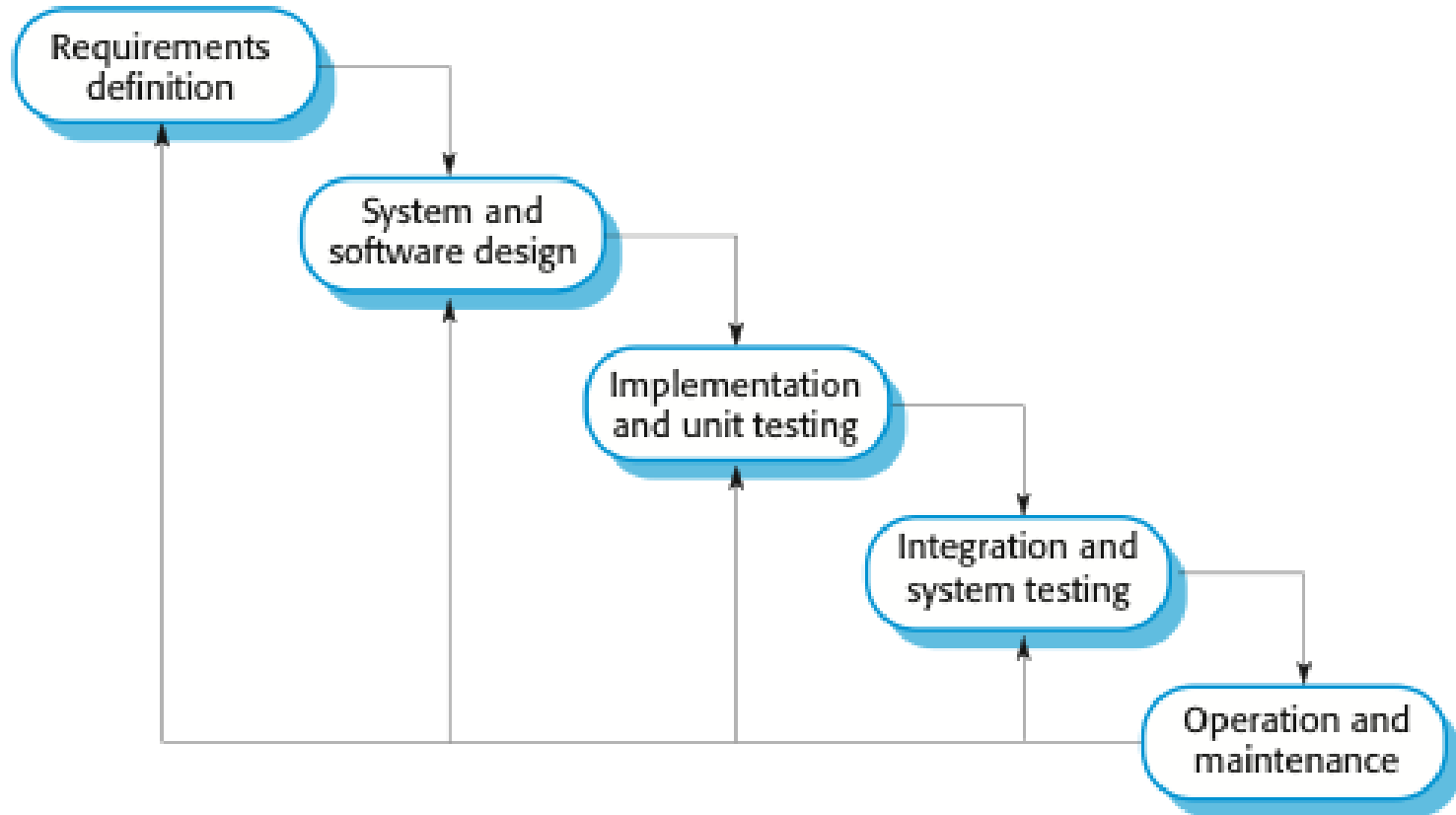
## ✧ Plan-driven development

- A plan-driven approach to software engineering is based around separate development stages with the **outputs** to be produced at each of these stages **planned in advance**.
- Not necessarily waterfall model – plan-driven, incremental development is possible

## ✧ Agile development

- Specification, design, implementation and testing are interleaved and the **outputs** from the development process are **decided through a process of negotiation** during the software development process.

# The waterfall model





# Waterfall model benefits and problems



- ✧ The waterfall model is mostly used for **large system engineering projects** where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.
- ✧ Suitable for new versions of **generic products**.
  - Well understood context, stable requirements.
- ✧ The process makes it difficult to respond to **changing customer requirements**.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes can be limited.

# Software prototyping



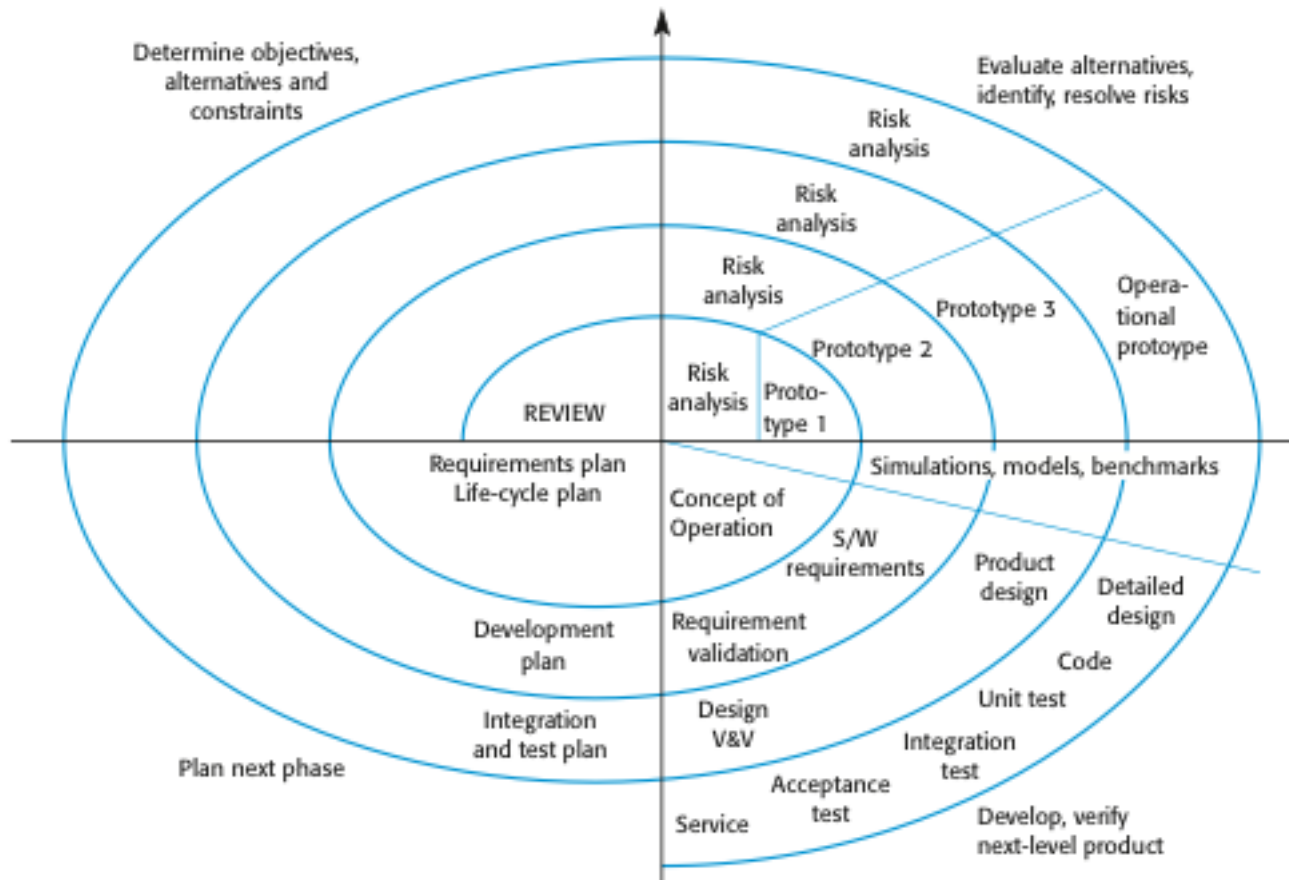
- ✧ A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- ✧ A prototype can be used in:
  - The requirements engineering process to help with **requirements elicitation**, consistency checking and validation;
  - In design processes to **explore design options** and develop a **UI design**;
- ✧ Prototypes often have **poor internal structure** and thus should not become the foundation of the final system.

# Boehm's spiral model



- ✧ **Process is represented as a spiral** rather than as a sequence of activities with backtracking.
- ✧ Each loop in the spiral represents a phase in the process.
- ✧ **No fixed phases** such as specification or design - loops in the spiral are chosen depending on what is required.
- ✧ **Risks** are explicitly assessed and resolved throughout the process.

# Boehm's spiral model of the software process



# Spiral model sectors



## ✧ Objective setting

- Specific objectives for the phase are identified.

## ✧ Risk assessment and reduction

- Risks are assessed and activities put in place to reduce the key risks.

## ✧ Development and validation

- A development model for the system is chosen which can be any of the generic models.

## ✧ Planning

- The project is reviewed and the next phase of the spiral is planned.

# The Rational Unified Process

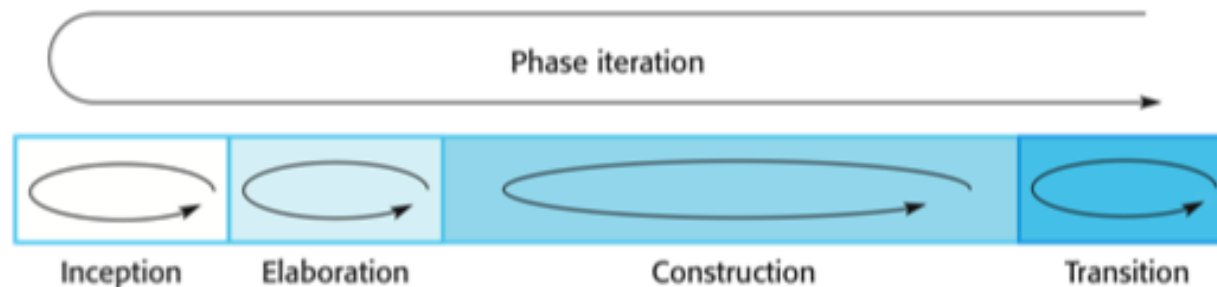


- ✧ A modern generic process commonly associated with the Unified Modeling Language (UML).
- ✧ Brings together aspects of a number of generic process models discussed in this lecture. Which ones?
- ✧ Normally described from 3 perspectives
  - A **dynamic perspective** that shows phases over time;
  - A **static perspective** that shows process activities;
  - A **practice perspective** that suggests good practices to be used during the process.

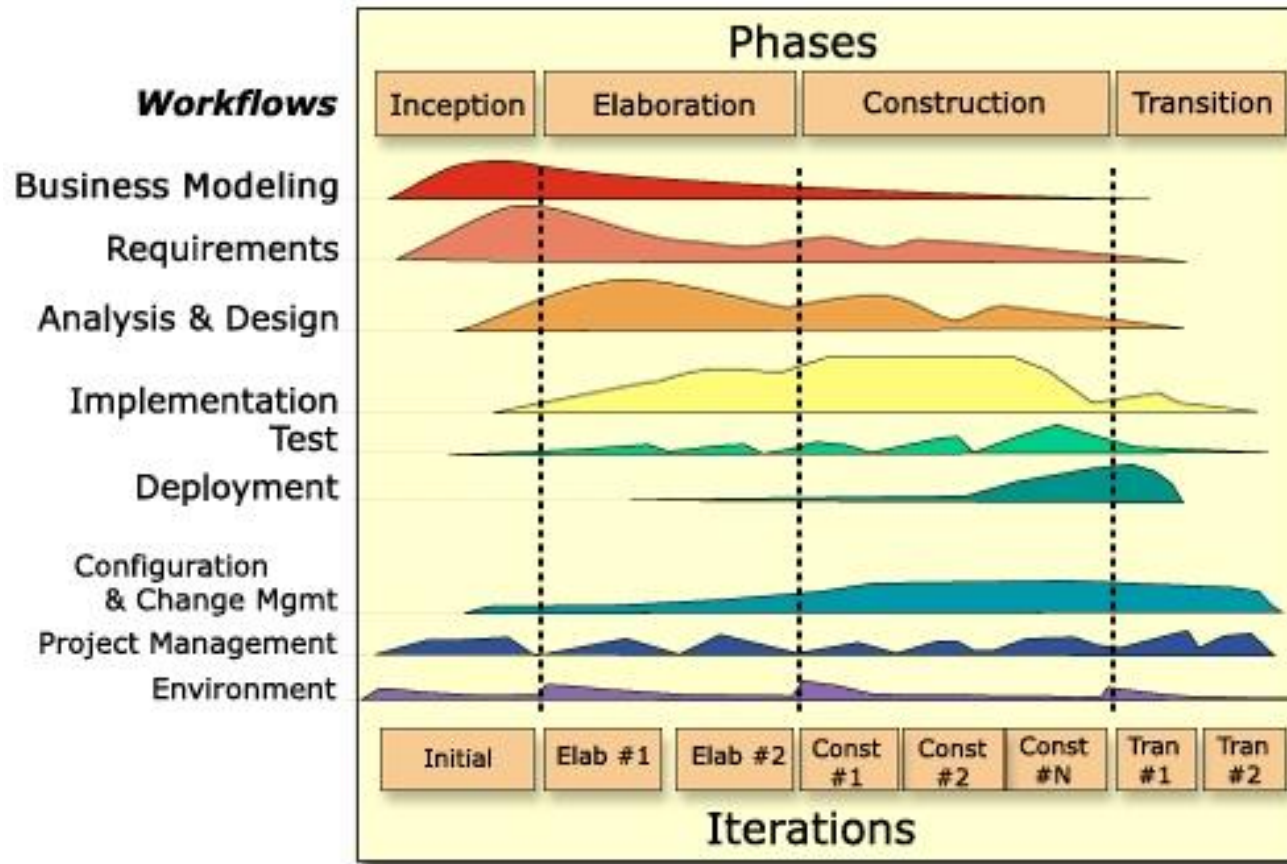
# Phases in the Rational Unified Process



- ✧ Inception
  - Establish the business case for the system.
- ✧ Elaboration
  - Develop understanding of the problem domain and system architecture.
- ✧ Construction
  - System design, programming and testing.
- ✧ Transition
  - Deploy the system in its operating environment.



# RUP process architecture

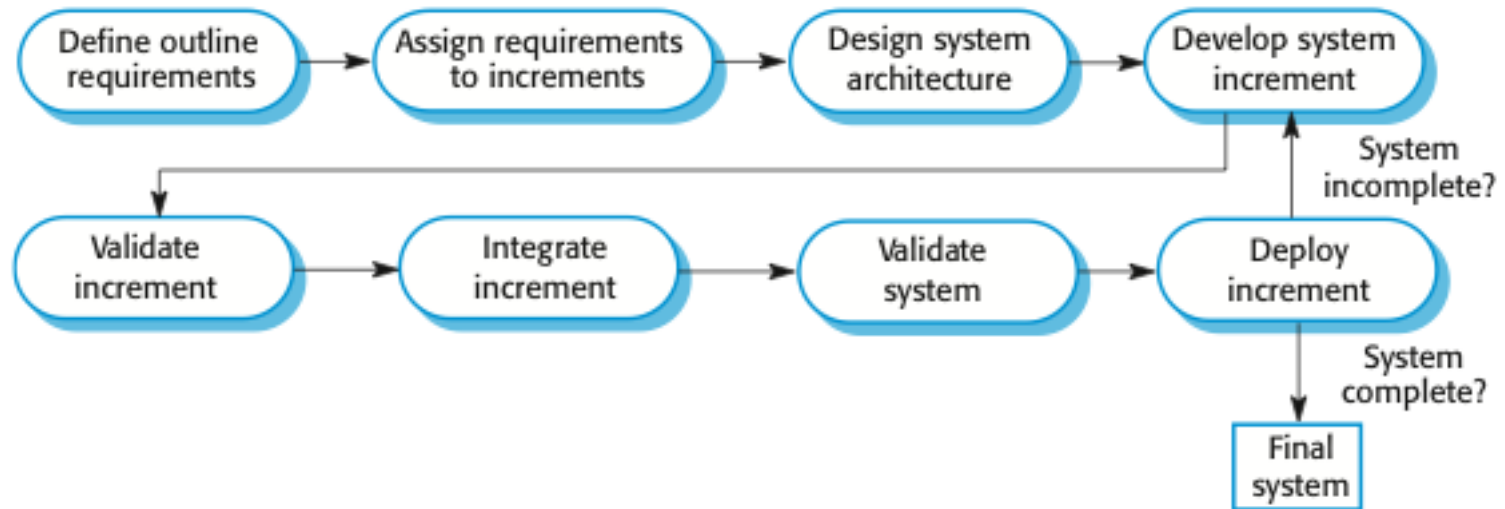




# Iterative and incremental development



✧ What is the difference between the two?



# Incremental delivery



- ✧ Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with **each increment delivering part of the required functionality**.
- ✧ User requirements are **prioritised** and the highest priority requirements are included in early increments.
- ✧ Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

# Incremental development benefits



- ✧ **Customer value** can be delivered with each increment so system functionality is available earlier.
- ✧ Early increments act as a **prototype** to help elicit requirements for later increments.
- ✧ **Lower risk** of overall project failure.
- ✧ The **highest priority** system services tend to receive the most attention (design, testing, etc.).

# Incremental development problems



- ✧ The **complete specification** is hard to foresee.
  - This becomes problematic when complete specification is required in contract negotiation.
- ✧ System **structure tends to degrade** as new increments are added.
  - Unless time and money is spent on extensive **refactoring**, regular changes tend to **corrupt system structure** and increase the cost of incorporating further changes.
- ✧ It is hard to identify and effectively design basic **facilities shared** by different parts of the system.
- ✧ The process is not visible, **progress is hard to trace**.

# Agile methods



## ✧ Agile methods:

- Focus on the **code** rather than the design
  - Are based on an **iterative approach** to software development
  - Are intended to deliver working software quickly and evolve this quickly to **meet changing requirements**.
- ✧ The aim of agile methods is to **reduce overheads in the software process** (e.g. by limiting documentation) and to be able to **respond quickly to changing requirements** without excessive rework.

# The principles of agile methods



Principle	Description
<b>Customer involvement</b>	Customers should be closely involved throughout the development process. Their role is provide and prioritize new requirements and to evaluate the iterations of the system.
<b>Incremental delivery</b>	The software is developed in increments with the customer specifying the requirements to be included in each increment.
<b>People not process</b>	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
<b>Embrace change</b>	Expect the system requirements to change and so design the system to accommodate these changes.
<b>Maintain simplicity</b>	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

# Problems with agile methods



- ✧ It can be difficult to **keep the interest of customers** who are involved in the process.
- ✧ Because of their focus on small, tightly-integrated teams, one needs to be careful when **scaling agile methods** to large systems.
- ✧ **Prioritizing changes** can be difficult where there are **multiple stakeholders**.
- ✧ Maintaining **simplicity** requires extra work.
- ✧ **Contracts** may be a problem as with other approaches to iterative development.

# Extreme programming



- ✧ Perhaps the best-known and most widely used agile method.
- ✧ Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.



# XP and agile principles



- ✧ Incremental development is supported through small, regular, **frequent system releases**.
- ✧ Customer involvement means **full-time customer engagement** with the team.
- ✧ People not process through **pair programming**, **collective ownership** and a process that avoids long working hours.
- ✧ Maintaining simplicity through **constant refactoring** of code.

# Reuse-oriented software engineering



- ✧ Based on **systematic reuse** where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- ✧ Process stages
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.
- ✧ Reuse is now the standard approach for building many types of business system

# Key points



- ✧ General process models describe the organization of software processes.
  - Examples of general models include the ‘waterfall’ model, incremental development, and reuse-oriented development.
- ✧ Processes should include activities to cope with change.
  - This may involve **prototyping** and **incremental delivery**, which help to avoid poor early decisions on requirements and design.
- ✧ Agile methods are incremental development methods that focus on frequent releases, reducing process overheads and emphasize customer involvement.



# Outline of Advanced Techniques

## Lecture 12/Part 3

# Software reuse



- ✧ In most engineering disciplines, systems are designed by composing existing components that have been used in other systems.
- ✧ Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need a design process that is based on systematic software reuse.
- ✧ There has been a major switch to **reuse-based development** and **Component-Based Development** over the past 10 years.

# Distributed systems



✧ Virtually all large computer-based systems are now distributed systems.

“... a collection of independent computers that appears to the user as a single coherent system.”

✧ Distributed systems issues

- Distributed systems are **more complex** than systems that run on a single processor.
- Complexity arises because different parts of the system are **independently managed** as is the network.
- There is **no single authority** in charge of the system so top-down control is impossible.

# Service-oriented architectures



- ✧ A means of developing distributed systems where the components are stand-alone services
- ✧ Services may execute on different computers from different service providers
- ✧ Standard protocols have been developed to support service communication and information exchange
- ✧ Benefits of SOA:
  - Services can be provided **locally** or **outsourced** to ext. providers
  - Services are **language-independent**
  - Investment in **legacy systems** can be preserved
  - Inter-organisational computing is facilitated through **simplified information exchange**

# Mobile applications



- ✧ A mobile applications include apps designed to run on smartphones, tablet computers and other mobile devices.
- ✧ They are usually available through application distribution platforms, operated by the owner of the mobile operating system, such as the Apple App Store, Google Play, and Windows Phone Store.
- ✧ Mobile apps were originally offered for general productivity and information retrieval, including email, calendar, contacts and weather information.
- ✧ However, public demand drove rapid expansion into many other categories, including banking, order-tracking, or medical apps.



# Embedded systems



- ✧ Computers are used to control a wide range of systems from **simple domestic machines**, through **games controllers**, to entire **manufacturing plants**.
- ✧ Their software must react to events generated by the hardware and, often, issue control signals in response to these events.
- ✧ The software in these systems is **embedded in system hardware**, often in **read-only memory**, and usually responds, in **real time**, to events from the system's environment.
- ✧ Issues of **safety** and **reliability** may dominate the system design.

# Cloud computing

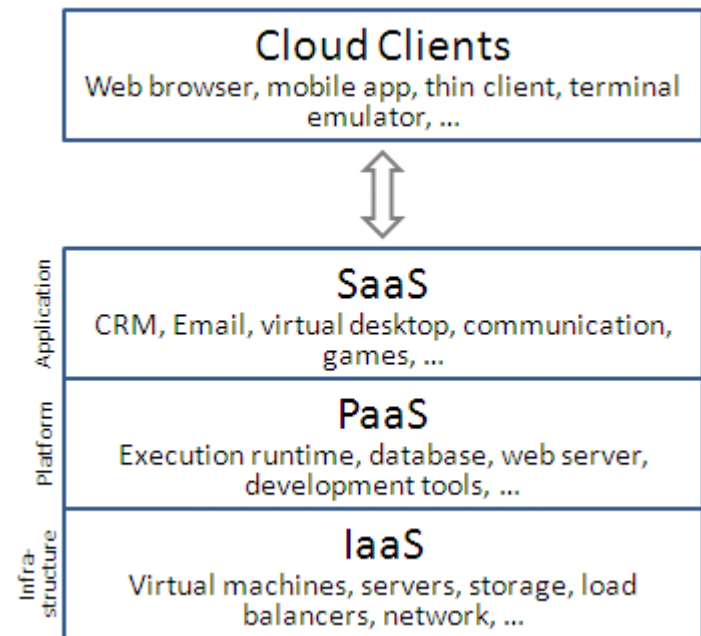


✧ Cloud computing is computing in which large groups of remote servers are networked to allow **centralized data storage** and online **access to computer services or resources**.

✧ Service models

- Infrastructure as a service (IaaS)
- Platform as a service (PaaS)
- Software as a service (SaaS)

✧ Moreover, **big data and its processing** is a topic on its own





# Tool Support

## Lecture 12/Part 4

# SE tasks commonly supported by tools



- ✧ Plan and schedule software development project
- ✧ Specify, manage and trace requirements
- ✧ Model and analyze business processes
- ✧ Create design and deployment models
- ✧ Create, edit, compile and debug code in different languages
- ✧ Generate and import database schema
- ✧ Track changes
- ✧ Manage tests
- ✧ Document software development
- ✧ Communicate and develop team based projects

# Most popular tools



- ✧ Requirements analysis and design modeling tools
- ✧ Programming environments that automate parts of program construction processes (e.g., automated builds)
- ✧ Software configuration management and version control
- ✧ Testing tools including static and dynamic analysis tools
- ✧ Continuous integration and release management
- ✧ Issue tracking
- ✧ Project management tools
- ✧ Tool integration concepts and mechanisms

# Integrated development environments (IDEs)



- ✧ Software development **tools are often grouped** to create an integrated development environment (IDE).
- ✧ An IDE is a set of software tools that supports different aspects of software development, within some **common framework** and user interface.
- ✧ IDEs are created to support development in a **specific programming language** such as Java. The language IDE may be developed specially, or may be an instantiation of a general-purpose IDE, with specific language-support tools.

# Key points



- ✧ Software engineering process can be supported by a large variety of tools.
- ✧ The specific tools are often integrated into a single environment or framework, which assists the developers through integrated support on one place.



# Course Follow-up

## Lecture 12/Part 5



# Course finalization



## ✧ Seminar projects

- Assessment
- “Seminar completion / Absolvování cvičení” notebook in IS

## ✧ Exam

- Number of exam dates
- Reservation/cancelation policies
- Length of the exam
- Form of the exam – test part and UML modelling part
- Results and their viewing

## ✧ Opinion poll

- Do not forget to give us your feedback! 😊

# Follow-up and related courses



- ✧ PA017 Softwarové inženýrství II
- ✧ PA103 Objektové metody návrhu informačních systémů
- ✧ PV167 Projekt z objektového návrhu inf. Systémů
- ✧ PV260 Software Quality
- ✧ PA104 Vedení týmového projektu
- ✧ PV207 Business Process Management
- ✧ PV165 Procesní řízení
- ✧ PV045 Management informačního systému
- ✧ PA189 Agile Management in IT
- ✧ PV028 Aplikační informační systémy

# Follow-up and related courses



- ✧ PV043 Informační systémy podniků
- ✧ PV230 Podnikové portály
- ✧ PV019 Geografické informační systémy I, II
- ✧ PV058 Informační systémy ve veřejné a státní správě
- ✧ PV213 Enterprise Information Systems in Practice
- ✧ PV098 Řízení implementace IS
- ✧ PB168 Základy databázových a informačních systémů
- ✧ PB114 Datové modelování I
- ✧ SSME Courses

# Thanks

---



Thank you for your attention  
and good luck with the exam!