Software processes

What is…

- Software process:
  - *set of **activities** that lead to production of software*
    - *Software specifications*
    - *Software design and implementation*
    - *Software validation*
    - *Software evolution*

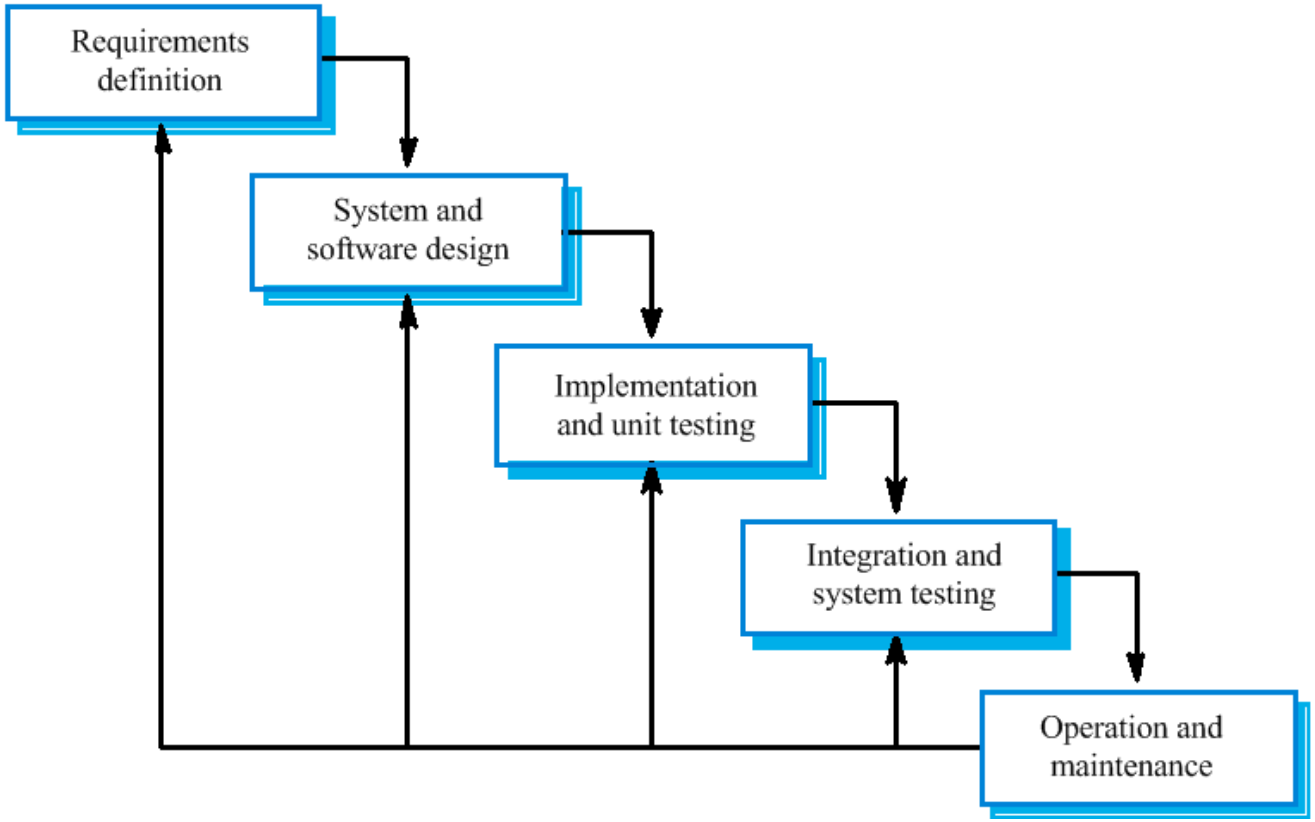
What is…

- Software process model:
  - abstract representation of a software process

- Paradigms (general models):
  - ❖Waterfall
  - ❖Incremental development
  - ❖Reuse-oriented SE

Waterfall model

- derived from system engineering processes



Activities

- Requirements definition:
  - services, constraints, goals are defined
  - serve as system specification

- System & software design
  - *System design*: partition the requirements (hardware / software), overall architecture
  - *Software design*: describe fundamental software system abstractions

Activities (2)

- Implementation & unit testing
  - write programs / program units
  - test each unit, check if it meets specifications
- Integration & system testing
  - Units are integrated & tested as a complete system

After testing, the software is delivered.

Activities (3)

- Operation and maintenance
  - Operation:
    - Install the system, put it to practical use
    - Train user
  - Maintenance:
    - Fix errors
    - Improve implementation
    - Adapt to new requirements

Waterfall model: products

- result of each phase:
  - one or more **documents** (expensive), *approved*
  - next phase can start only after the current phase has finished
- there is a certain overlapping
  - problems in phase *p* is identified in phase *p*+1
  - *each activity is executed as a sequence of iterations*
  - after some iterations, parts of development are frozen

Waterfall model: conclusions

- Advantages:
  - good documentation
  - agrees with other engineering process models
- Disadvantages
  - inflexible partitioning of the project into distinct phases
  - difficult to change requirements
  - requires commitment to what has been signed

Incremental development

→ Develop an initial implementation

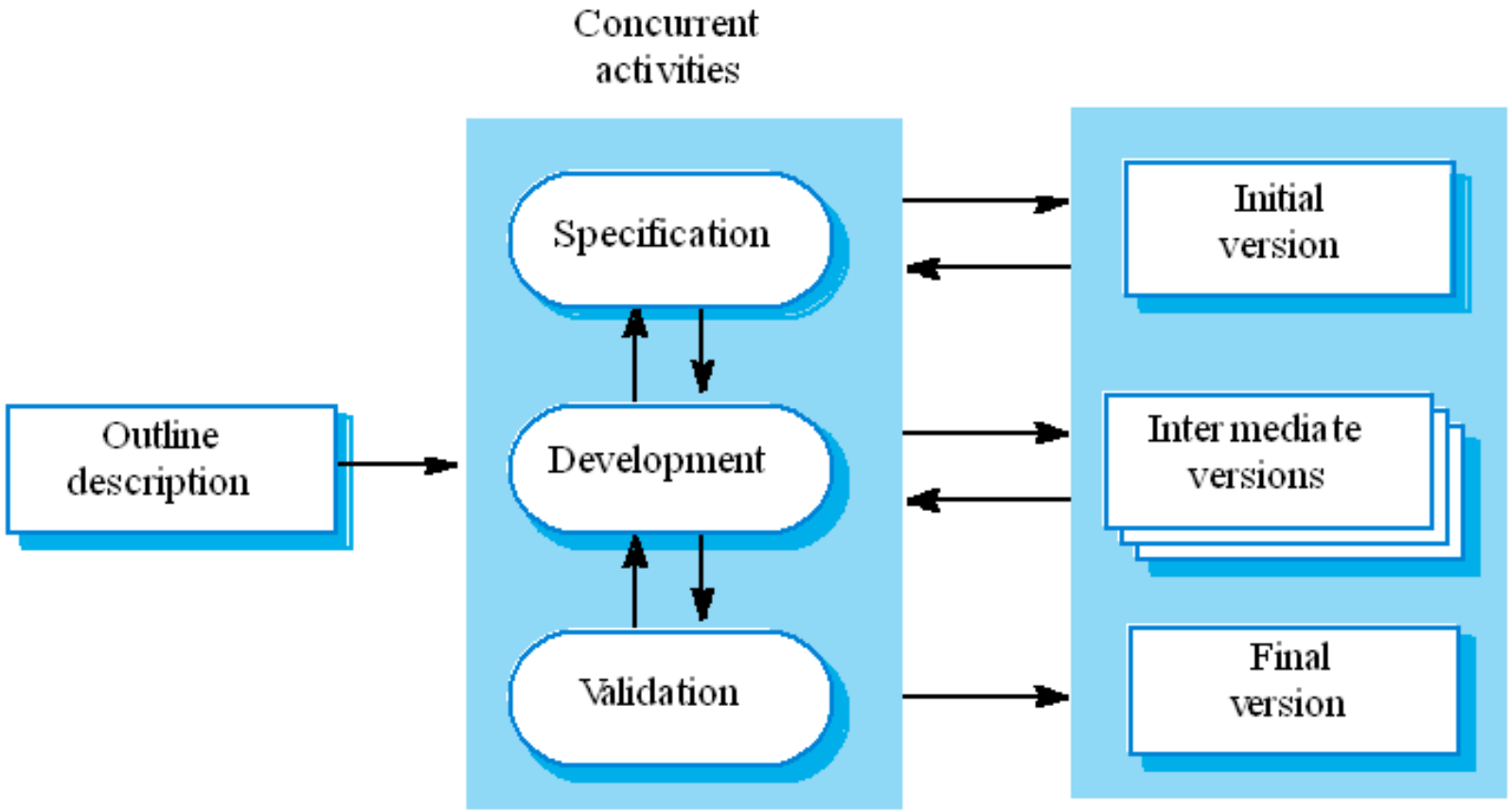
→ Expose to the user

→ Refine until an adequate system is obtained

Incremental development principles

- work with the customer
- start with parts that are understood
- evolve with new features proposed by customer

Incremental development



Incremental development: conclusions

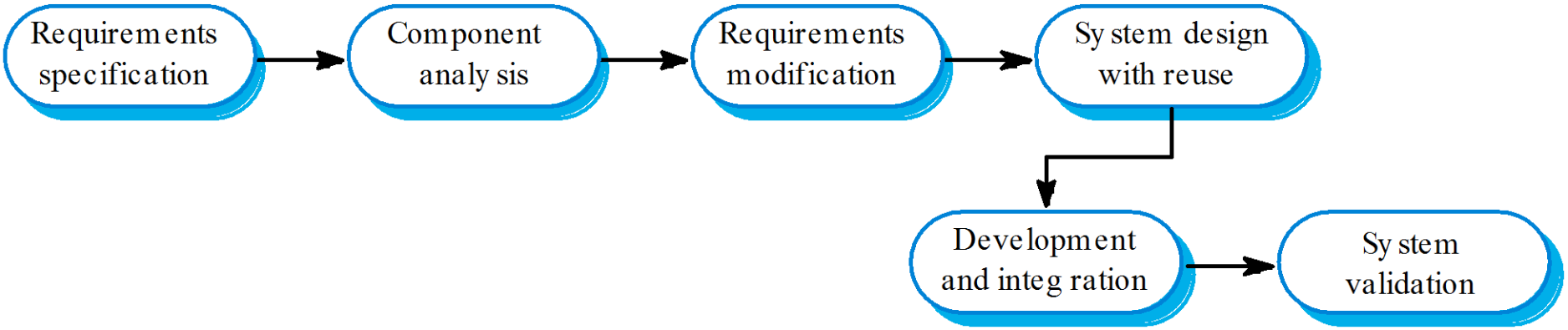
- Advantages:
  - Specifications can develop incrementally
  - Systems better tailored to the needs of customers than in *waterfall* approach
- Disadvantages:
  - The evolution of the process is not visible
  - Problems with documentation
    - Poor documentation, but cheap or
    - Extensive documentation, very expensive
  - Poorly structured systems

Reuse-oriented SE

- Reuse of existing software
  - Find software similar to what is needed
  - Modify it
  - Incorporate it into the system under development

- *There are lots of reusable software components*
- *There are frameworks for integrating them*

Reuse-oriented SE



Reuse-oriented SE: conclusions

- Advantages:
  - Reduced effort for developing own software
  - Faster delivery
- Disadvantages
  - Requirements compromises
  - *System may go astray from what the user needs*
  - Most components are not developed in-house

    →No control of the specs of new versions

Coping with Changes

- requirements change
- priorities change
- technologies change

- freezing specs & design may lead to outdated software
- iterative processes:
  - *specs are developed along with the software*

Coping with Changes

- Approaches:
  - Avoid change
    - Anticipate deviations before committing
  - Tolerate change
    - design the process to easily integrate changes
- Practices
  - System prototyping
  - Incremental delivery

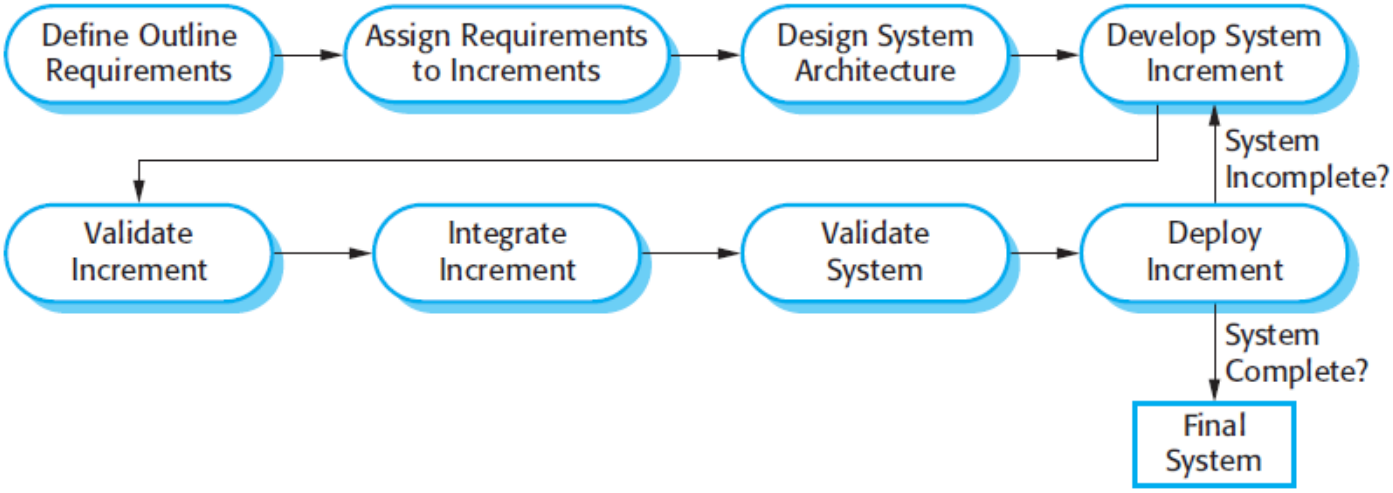
System prototyping

- Develop quickly a simplified version
- Allow the customer to experiment with it
  - ➢ check customer's requirements
  - ➢ validate design decisions
- Benefits
  - – users play with the prototype before delivery of the whole system
    - ➢ fewer requirements changes after delivery

→ Change avoidance

Incremental delivery

- System features– *split into increments, delivered in turns*



- Priorities are assigned to requirements
- Increments provide services with highest priorities first

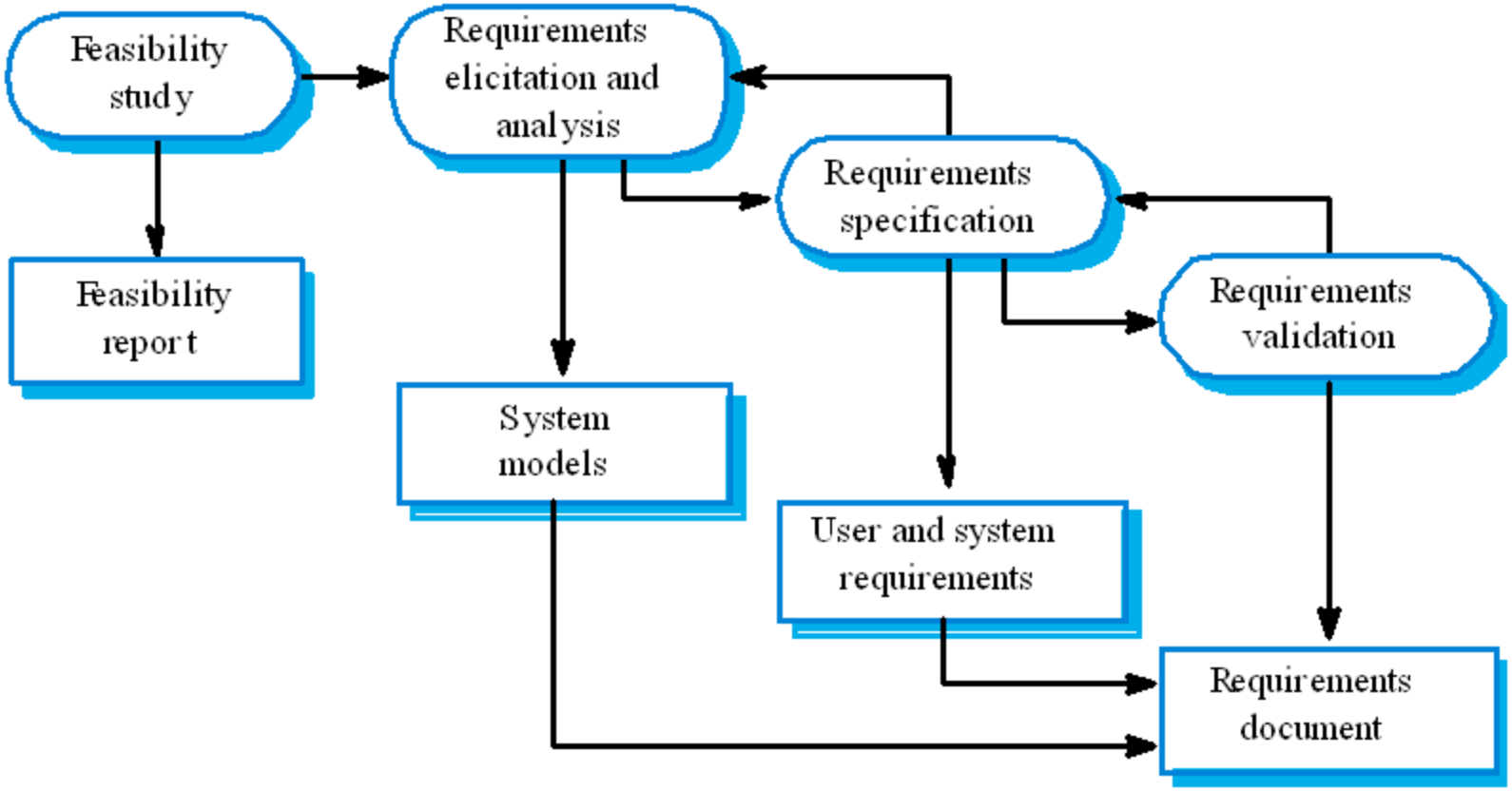
Incremental delivery: conclusions

- **Advantages**:
  - Customers get something useful earlier
  - By using delivered systems (even incomplete), customers gain insight
  - Low risk of overall failure
  - Most important services get most testing

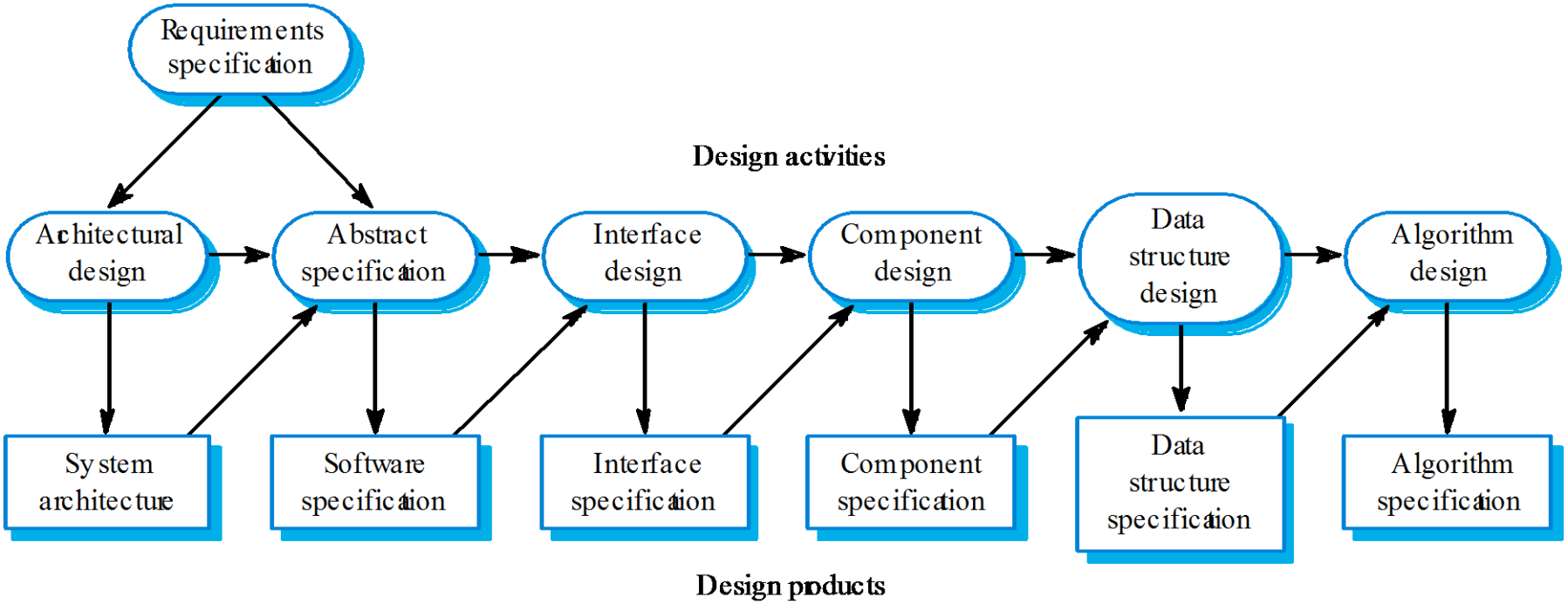
  → **Change avoidance and tolerance**
- **Disadvantages**:
  - Increments should have similar “size”
  - Hard to see from the beginning common facilities needed by all parts of the system

Activities in software processes

Activities: Requirements



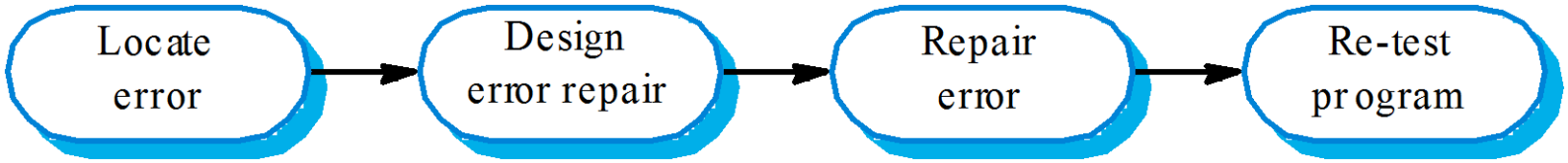
Activities: Design



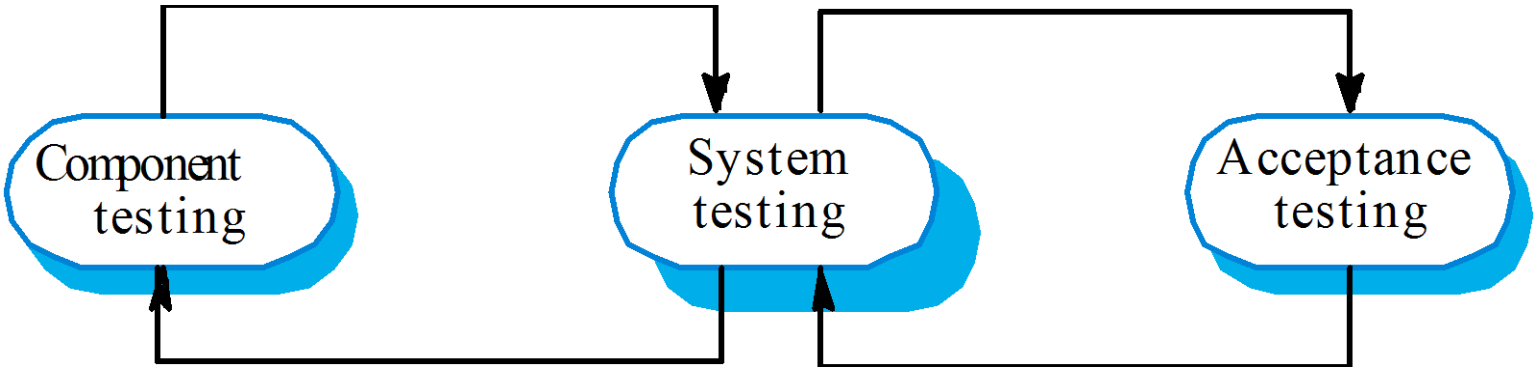
Activities: Implementation

translate the output
of the design activities
into code

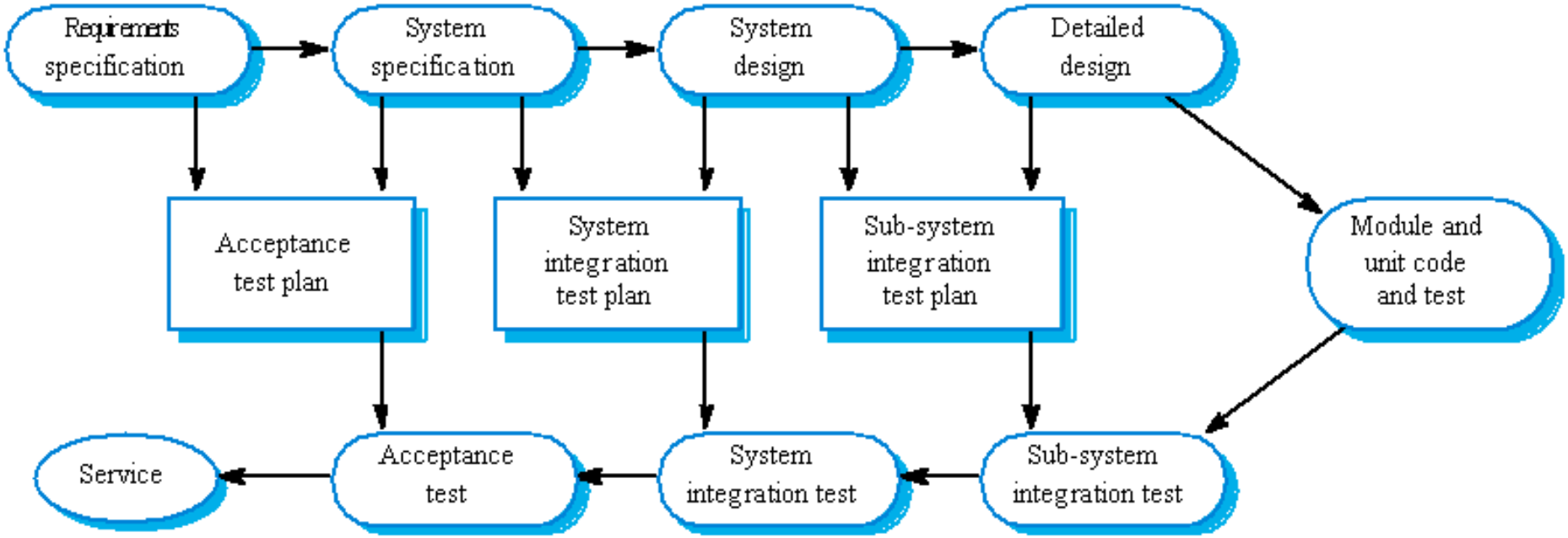
Activities: Debugging



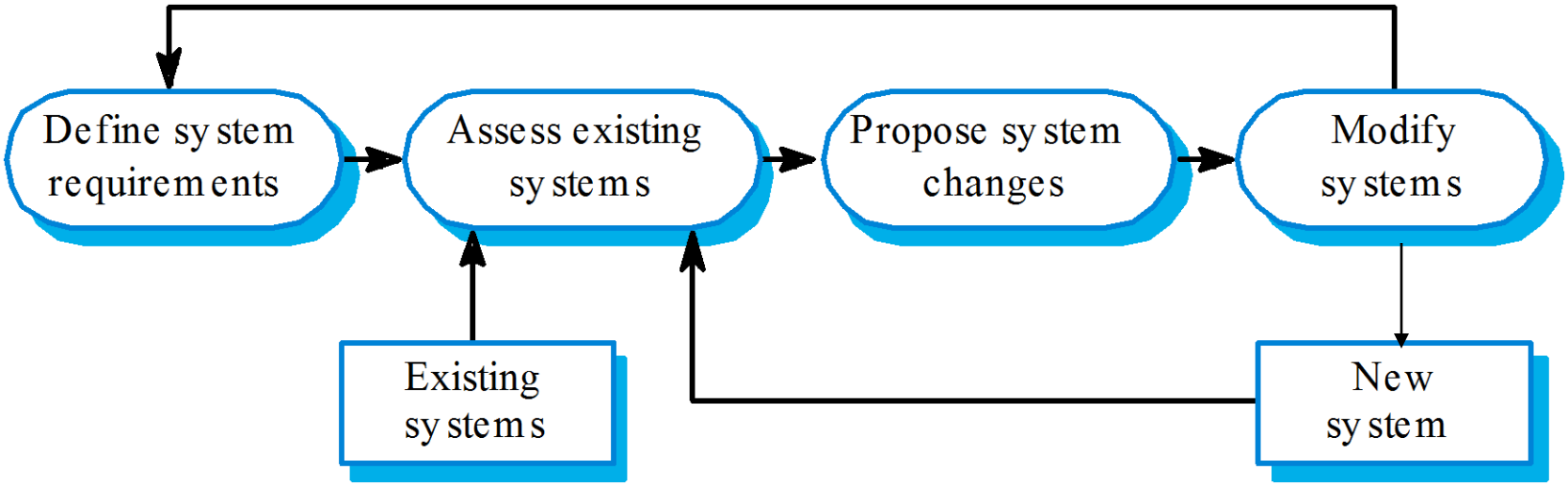
Activities: Validation & Verification



Activities: V&V – Testing



Software evolution



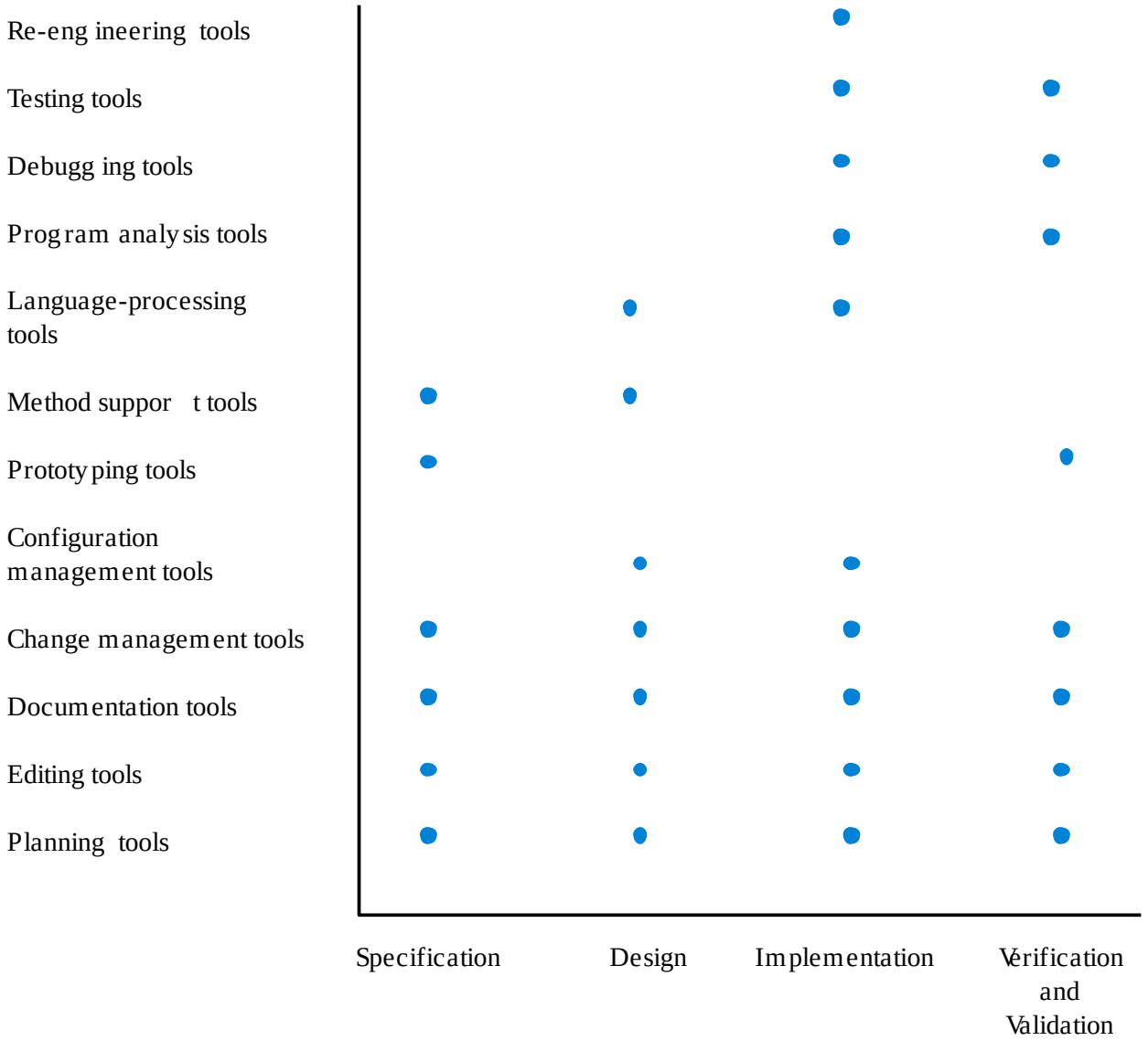
CASE

- Software that supports software process activities
- Types:
  - Tools
  - Workbenches
  - Frameworks

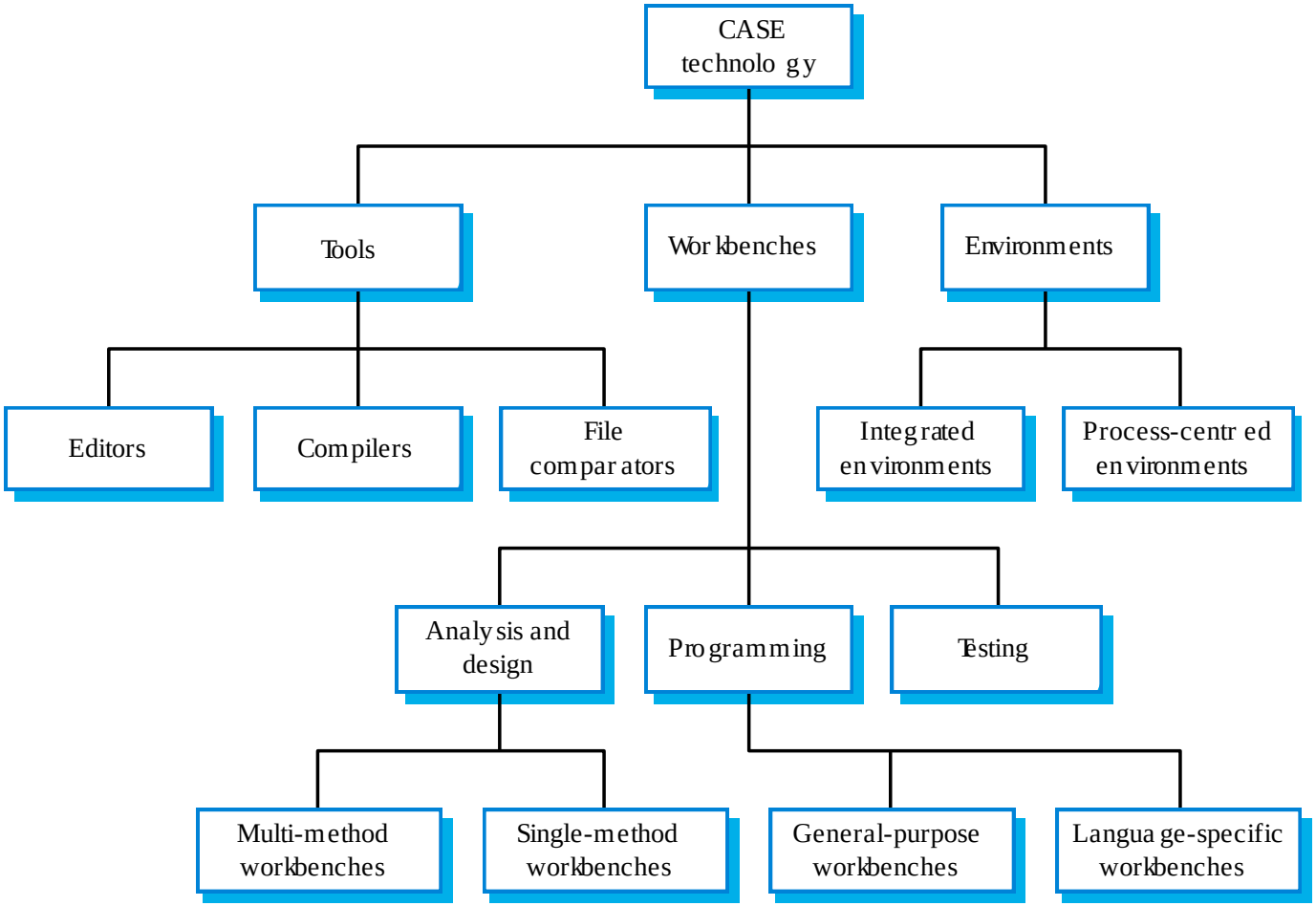
CASE tools

Tool type	Examples
Planning tools	PERT tools, estimation tools, spreadsheets
Editing tools	Text editors, diagram editors, word processors
Change management tools	Requirements traceability tools, change control systems
Configuration management tools	Version management systems, system building tools
Prototyping tools	Very high-level languages, user interface generators
Method-support tools	Design editors, data dictionaries, code generators
Language-processing tools	Compilers, interpreters
Program analysis tools	Cross reference generators, static analysers, dynamic analysers
Testing tools	Test data generators, file comparators
Debugging tools	Interactive debugging systems
Documentation tools	Page layout programs, image editors
Re-engineering tools	Cross-reference systems, program re-structuring systems

CASE tools


Re-eng ineering tools
Testing tools
Debugg ing tools
Program analysis tools
Language-processing tools
Method suppor t tools
Prototyping tools
Configuration management tools
Change management tools
Documentation tools
Editing tools
Planning tools
Specification
Design
Implementation
Verification and Validation

CASE tools


CASE technology
Tools
Workbenches
Environments
Editors
Compilers
File comparators
Integrated environments
Process-centred environments
Analysis and design
Programming
Testing
Multi-method workbenches
Single-method workbenches
General-purpose workbenches
Language-specific workbenches

C#

Ms Visual C# 201x Express

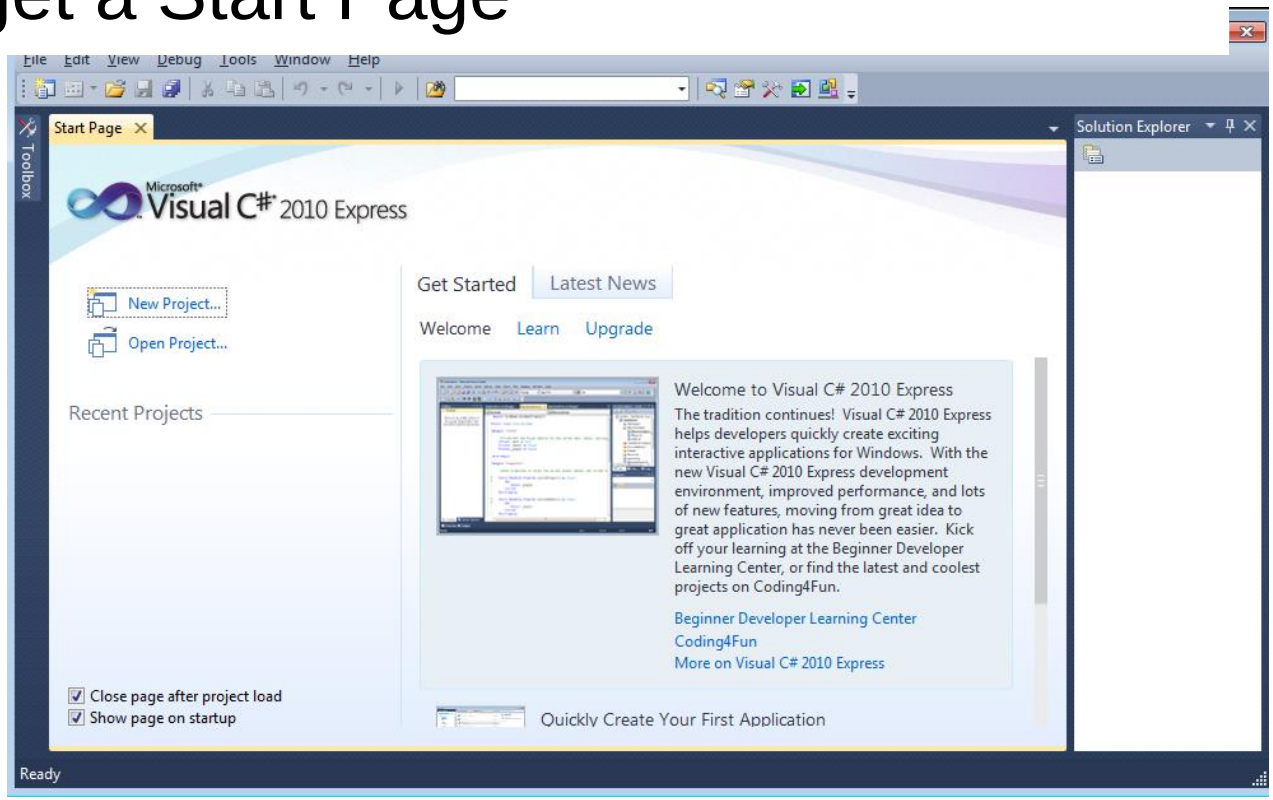
- Download e.g. from

  https://www.visualstudio.com/thank-you-downloading-visual-studio/?sku=Community&rel=15

C# / MVS

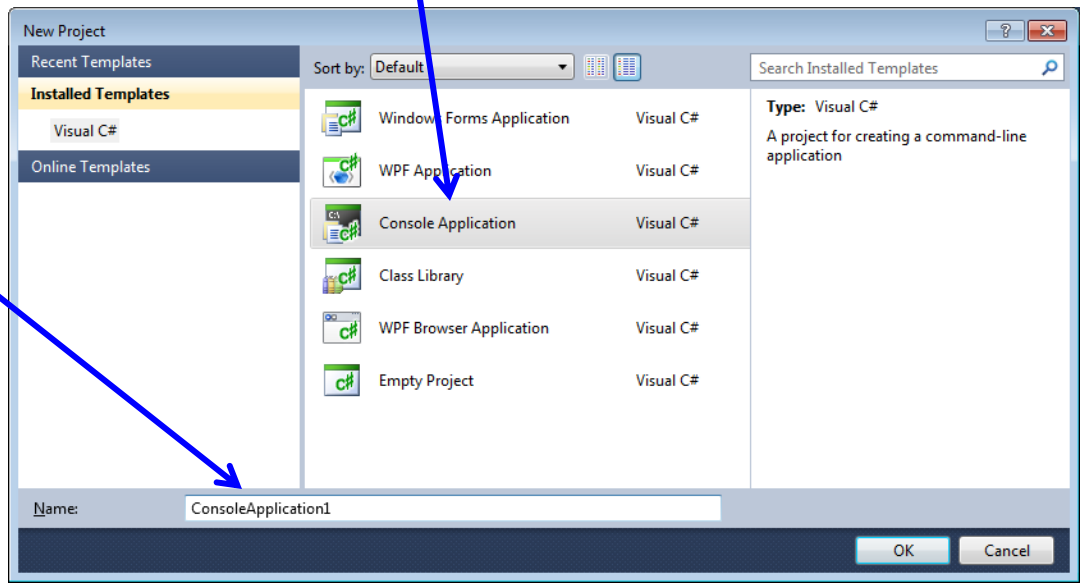
- Start Ms Visual C# 201* Community Edition
  - . . . and be patient.

  →you shall get a Start Page



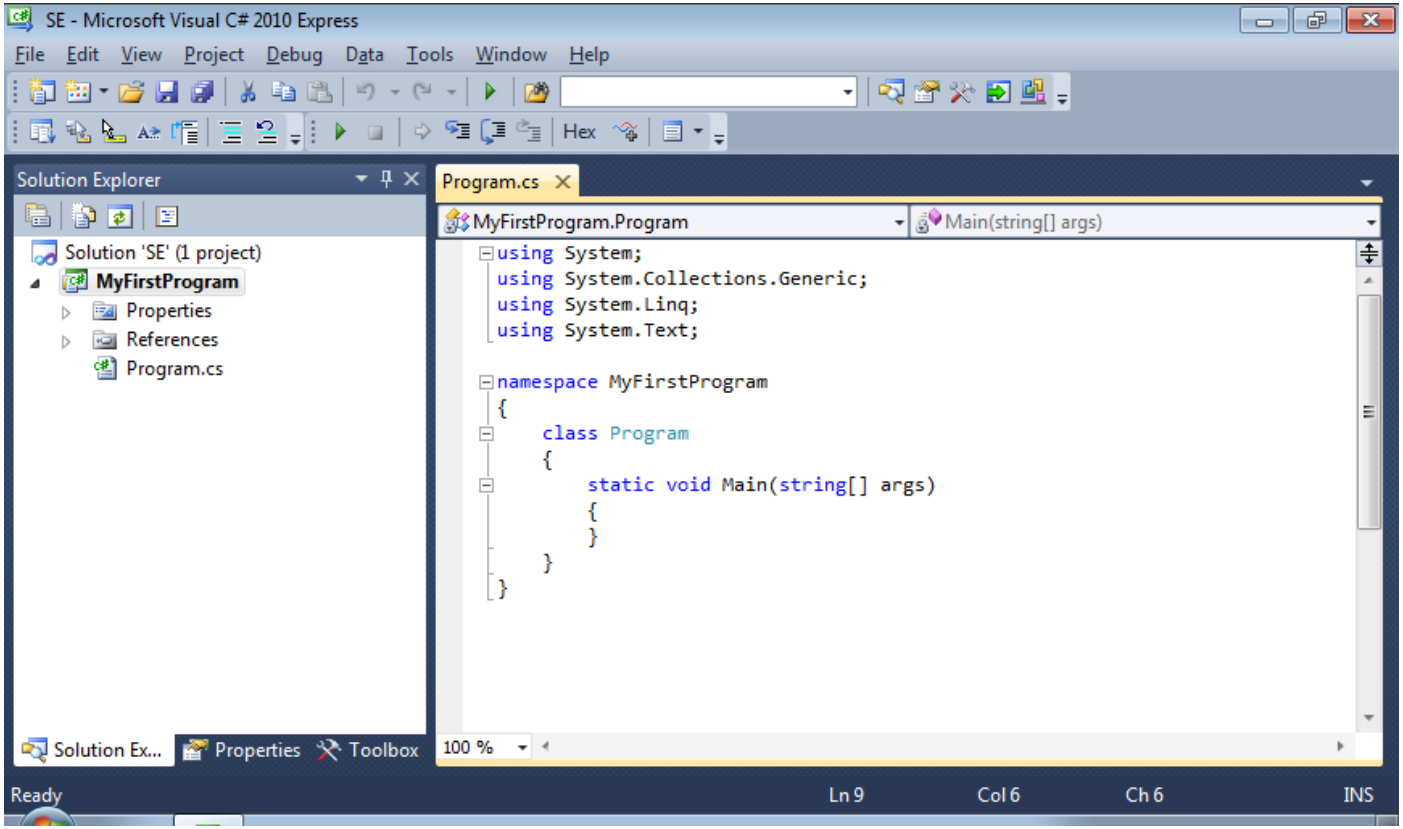
First project

- Click on “New Project”
- Select “Console Application” from the list of possible project types
- Give a reasonable name to the new project



Environment visual settings

- Arrange necessary views to get your favorite environment configuration, e.g.:



Simple “Hello World” project

- Add the following line to the “Main” function:

```
Console.WriteLine("Hello World!");
```

- Run the project, by clicking on the usual button.

  → you will not see much. Why?

The HelloWorld program

```
using System;

namespace MyFirstProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

C# fundamentals

- C# program
  - one or more files
  - .cs files contain C# code
    - “using” directives – to import types defined in other namespaces
    - 0 or more namespaces, containing:
      - inner namespaces
      - classes
      - structs
      - enums
      - delegates
    - the last four concepts are considered *type definitions*.

.NET [C#] fundamentals

- After compilation, *assemblies* are generated
  - executable (.exe), or
  - dynamic-link libraries (DLLs, .dll).
- assemblies contain special assembly code
  - MSIL – Ms Intermediate Language
- native code is generated at execution time
  - by a just-in-time (JIT) compiler

C# fundamentals (continued)

- Everything must be inside a class!
  - there are no stand-alone functions,
  - there are no global data.

  *some **object-oriented programming** concepts:*
  - *objects: defined by **state** and **behaviour***
  - *classes: blueprints for objects*
    → *objects – instances of classes*
  - *inheritance: from general to specialized classes*
  - *interfaces: common behaviour aspects*
    → *this is new!*

C# classes

- Concepts: (similar to C++)
  - Class
  - Instances of the class
  - Member
    - Data → *states*
    - Functions → *behaviour*
  - Class members, instance members
  - Access modifiers (public, internal, protected…)

For individual study…

- C# documentation at MSDN
- C# programming guide
- General structure of a C# program

- C# language specifications

Homework

- What kind of software process (waterfall, exploratory, component-based, iterative, etc.) is recommended for each of the following projects?
  1. *Client*: OeBB.
     Problem: Optimal assignment of locomotive duties to locomotive drivers.
     Circumstances: The requirements are quite well known by the OeBB and comprehensible to you: The specifications are not expected to change for quite a few years. OeBB has a good idea about what they want. The project needs to be extensively documented.
  2. *Client*: A company that produces sensors for railroads.
     Problem: Graphical editor for railroad tracks in railroad stations.
     Circumstances: The company needs a basic editor pretty fast. In later phases, additional features are to be implemented.
  3. *Client*: A taxi company.
     Problem: An app for ordering taxis from mobile devices.
     Circumstances: The taxi company has no clear idea about how this app should run.