The Design Document

Design phase: deliverables

- Main delivery: **design document**

*Design = an activity that gives **structure** to the solution to a given problem*

- the design phase starts with the *requirements document* and maps the requirements into *architecture*
- the architecture defines the *components*, their *interfaces* and *behavior*
- the design document describes a *plan to implement the requirements*
- contains details on:
  - computer programming languages and environments,
  - machines,
  - packages,
  - application architecture,
  - distributed architecture layering, memory size, platform, algorithms, data structures, global type definitions, interfaces…
- may include the usage of existing components

Design document template (1)

- Introduction
- System Overview
- Design Considerations
  - Assumptions and Dependencies
  - General Constraints
  - Goals and Guidelines
  - Development Methods
- Architectural Strategies
  - *strategy-1 name or description*
  - *strategy-2 name or description*
  - ...
- System Architecture
  - *component-1 name or description*
  - *component-2 name or description*
  - ...
- Policies and Tactics
  - *policy/tactic-1 name or description*
  - *policy/tactic-2 name or description*
  - ...
- Detailed System Design
  - *module-1 name or description*
  - *module-2 name or description*
  - ...
- Glossary
- Bibliography

(Software projects survival guide)

Design document template (2)

- **Introduction**
  - Describe the purpose, scope and intended audience
  - Identify the system/product using any applicable names and/or version numbers.
  - Provide references for any other pertinent documents such as:
    - Related and/or companion documents
    - Prerequisite documents
    - Documents which provide background and/or context for this document
    - Documents that result from this document (e.g. a test plan or a development plan)
  - Define any important terms, acronyms, or abbreviations
  - Summarize (or give an abstract for) the contents of this document.

Design document template (3)

- **System Overview**
  - Provide a general description of the software system:
    - functionality and
    - matters related to the overall system and its design
    - [discussion of the basic design approach or organization]
- **Design Considerations**
  - describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution
- **Assumptions and Dependencies**
  - Describe any assumptions or dependencies regarding the software and its use:
    - Related software or hardware
    - Operating systems
    - End-user characteristics
    - Possible and/or probable changes in functionality

Design document template (4)

- **General Constraints**
  - global limitations or constraints that have a significant impact on the design:
    - Hardware or software environment
    - End-user environment
    - Availability or volatility of resources
    - Standards compliance
    - Interoperability requirements
    - Interface/protocol requirements
    - Data repository and distribution requirements
    - Security requirements (or other such regulations)
    - Memory and other capacity limitations
    - Performance requirements
    - Network communications
    - Verification and validation requirements (testing)
    - Other means of addressing quality goals
    - Other requirements described in the requirements specification

Design document template (5)

- **Goals and Guidelines**
  - goals, guidelines, principles, or priorities which dominate or embody the design of the system's software:
    - emphasis on speed versus memory use
    - working, looking, or "feeling" like an existing product
  - for each such goal or guideline, unless it is implicitly obvious, describe the reason for its desirability
- **Development Methods**
  - describe the method or approach used for this software design
  - include a reference to a more detailed description of formal or published methods

Design document template (6)

- **Architectural Strategies**
  - decisions and/or strategies that affect the overall organization of the system and its higher-level structures
  - should provide insight into the key abstractions and mechanisms used in the system architecture
  - the reasoning employed for each decision and/or strategy and how any design goals or priorities were balanced or traded-off
    - Use of a particular type of product (programming language, database, library, etc. ...)
    - Reuse of existing software components to implement various parts/features of the system
    - Future plans for extending or enhancing the software
    - User interface paradigms (or system input and output models)
    - Hardware and/or software interface paradigms
    - Error detection and recovery
    - Memory management policies
    - External databases and/or data storage management and persistence
    - Distributed data or control over a network
    - Generalized approaches to control
    - Concurrency and synchronization
    - Communication mechanisms
    - Management of other resources

Design document template (7)

- **System Architecture**
  - high-level overview of how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components
  - not too much detail about the individual components themselves
  - main purpose: to gain a general understanding of how and why the system was decomposed, and how the individual parts work together to provide the desired functionality
  - major responsibilities that the software must undertake and the various roles that the system (or portions of the system) must play
  - how the system was broken down into its components/subsystems
  - how the higher-level components collaborate with each other
  - provide some sort of rationale for choosing this particular decomposition
  - make use of design patterns
  - include any diagrams, models, flowcharts, documented scenarios or use-cases of the system behavior and/or structure
- **Subsystem Architecture**
  - more detailed discussion of particular components
  - how the component was further divided into subcomponents, and the relationships and interactions between the subcomponents
  - recurse if necessary, but leave the details for the *Detailed System Design* section

Design document template (8)

- **Policies and Tactics**
  - Choice of which specific product to use (compiler, interpreter, database, library, etc. ...)
  - Engineering trade-offs
  - Coding guidelines and conventions
  - The protocol of one or more subsystems, modules, or subroutines
  - The choice of a particular algorithm or programming idiom (design pattern) to implement portions of the system's functionality
  - Plans for ensuring requirements traceability
  - Plans for testing the software
  - Plans for maintaining the software
  - Interfaces for end-users, software, hardware, and communications
  - Hierarchical organization of the source code into its physical components (files and directories).
  - How to build and/or generate the system's deliverables (how to compile, link, load, etc. ...)

Design document template (9)

- **Detailed System Design**
  - detailed description of the components introduced in “System Architecture” chapter
  - *Classification*
    - kind of component, such as a subsystem, module, class, package, function, file, etc. ....
  - *Definition*
    - specific purpose and semantic meaning of the component.
  - *Responsibilities*
    - primary responsibilities and/or behavior of this component:
      - What does this component accomplish? What roles does it play?
      - What kinds of services does it provide to its clients?
  - *Constraints*
    - relevant assumptions, limitations, or constraints for this component: on timing, storage, or state
    - might include rules for interacting with this component (preconditions, postconditions, invariants, data formats and data access, synchronization, exceptions, etc.)
  - *Composition*
    - description of the use and meaning of the subcomponents that are a part of this component.

Design document template (10)

- **Detailed System Design** *(continuation)*
  - *Uses/Interactions*
    - collaborations with other components:
      - What other components is this entity used by?
      - What other components does this entity use?
    - known or anticipated subclasses, superclasses.
  - *Resources*
    - resources that are managed, affected, or needed by this entity: memory, processors, printers, databases, or a software library
    - discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.
  - *Processing*
    - how this components goes about performing the duties necessary to fulfill its responsibilities
    - encompass a description of any algorithms used; changes of state; relevant time or space complexity; concurrency; methods of creation, initialization, and cleanup; and handling of exceptional conditions.
  - *Interface/Exports*
    - services (resources, data, types, constants, subroutines, and exceptions) provided by this component

Design document template (11)

- **Detailed Subsystem Design**
  - detailed description of this software component (or a reference to such a description)
  - include diagrams showing the details of component structure, behavior, or information/control flow
- **Glossary**
  - ordered list of defined terms and concepts used throughout the document.
- **Bibliography**
  - list of referenced and/or related publications.

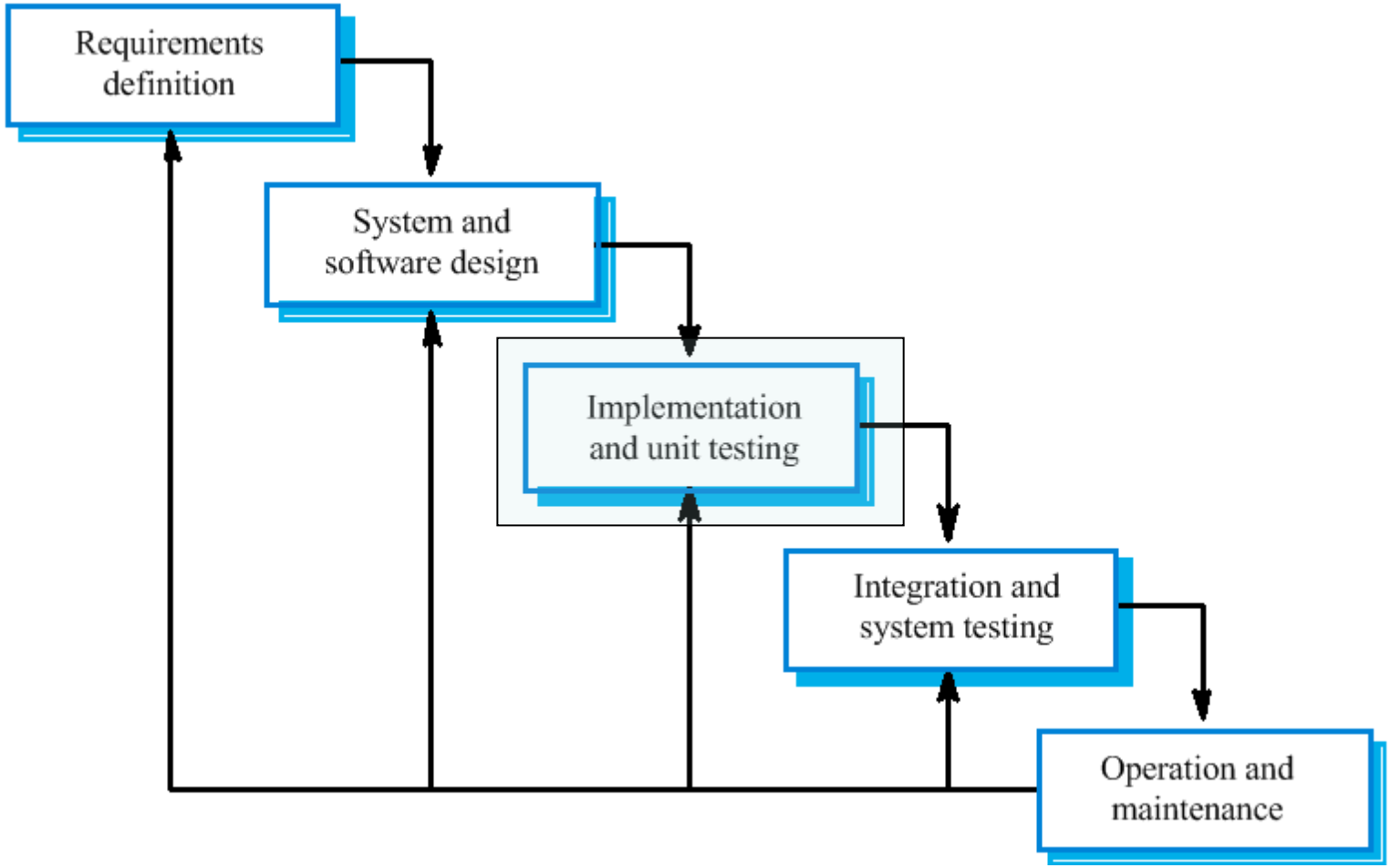
Development

Development

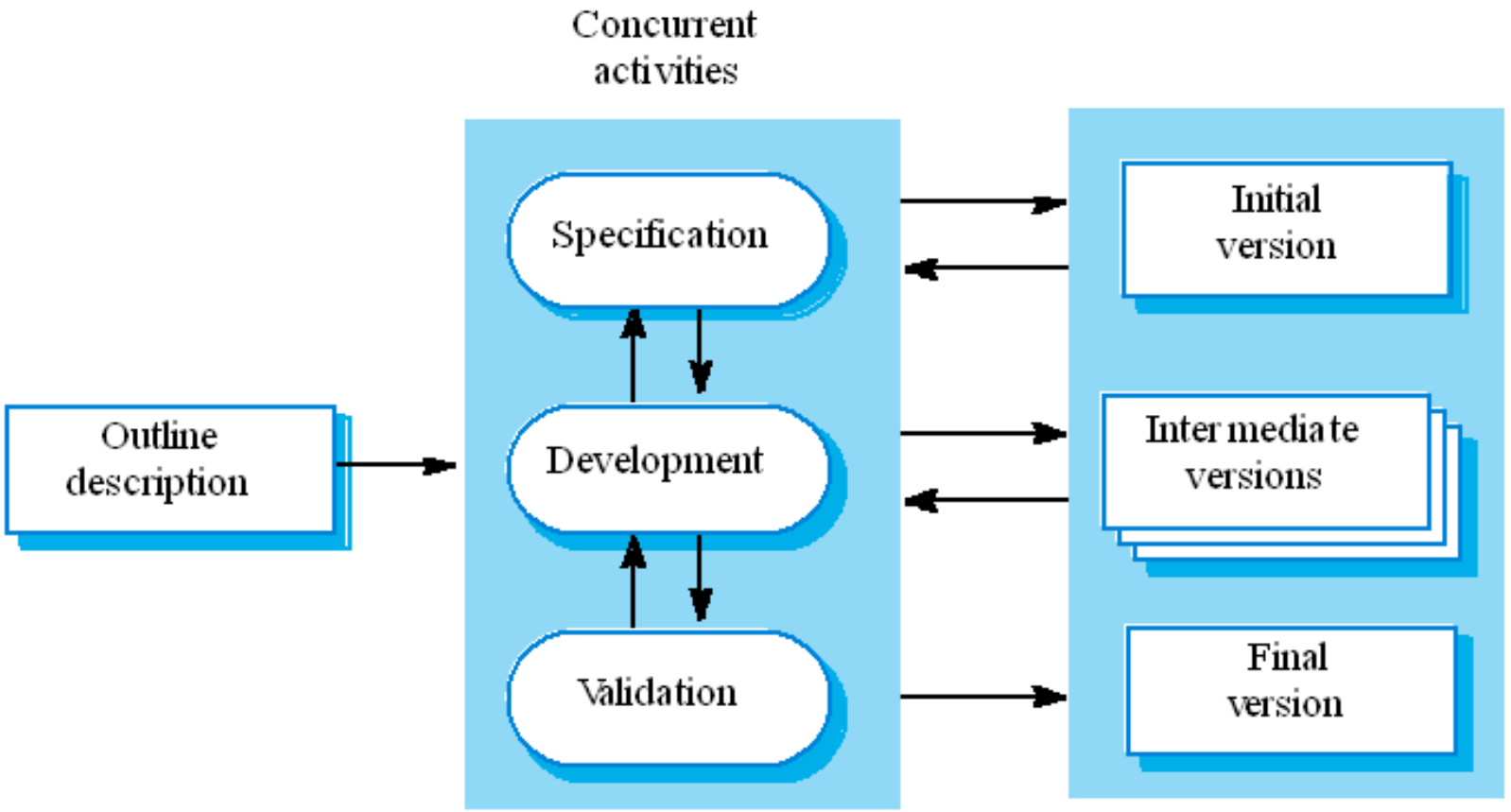
Development = writing a program based on a design specification

- Types of development
  - *Traditional* – waterfall model - coding starts after the system is fully specified and models have been designed
  - *Incremental* – produce & deliver software in increments

Traditional development



Incremental development



Coding

Coding

- Coding: the process of writing programs
- Coding conventions
  - Set of rules that guide the shape of written code
  - Important for improving the readability and understandability
  - almost always company-specific
    - each software company has its own guidelines and conventions for writing code

Code conventions

- File organization
- Naming conventions
- Formatting
- Statements and declarations

File organization

- A file should contain one class
- There should be a specific order:
  - Beginning comments
  - [file guard – for header files in C++]
  - Include / import / using
  - Class declarations / definitions

File structure conventions

- Beginning comments:

```
/*****************************************************************
 * File                 : Graph.cs
 * Classes              : Graph
 * Namespaces           : CombinatorialOptimization.GraphBase
 * Author               : Petru Pau
 * Initial author       : Petru Pau
 * Date                 : 24 May 2006
 * Copyright(c)         : 2006 RISC Software GmbH
 *****************************************************************
 *
 * Description: class Graph contains the abstract data type
 *    graph: A collection of nodes and edges.
 *
 *****************************************************************/
```

File structure conventions (2)

- File guards (for C++ header files)

```
#ifndef FILENAME_H
#define FILENAME_H
. . .
#endif // FILENAME_H
```

File structure conventions

- Class declarations / definitions; example:
  - Public methods
  - Protected methods
  - Private methods

  - [Public variables] – should not exist
  - Protected variables
  - Private variables

Naming conventions

- Depend on the programming language
- Examples:
  - Descriptive names: meaningful, self-explanatory, in English
  - Avoid abbreviations (unless necessary: `URL`)
  - Positive meaning
    - `isEmpty()`, not `isNotEmpty()`,
    - `isEnabled()` not `isDisabled()`)
  - Differentiate individual words by capitalizing: `shortestPath`, not `shortest_path`
  - Class names capitalized

Style conventions

- Lines:
  - Not too long (max 120 characters)
  - One statement per line
  - If breaks are necessary:
    - After a comma
    - Before an operator
    - Align the new line
    - Indent
  - Align code sequences with similar structure

Style conventions

- Methods:
  - Not too long (max 25 lines)
    - If longer, split into more methods
  - Single-purpose
  - Not too many parameters
  - Avoid side-effects

Style conventions

- Document the code:
  - Describe each class
  - Describe methods
  - Describe statements (trailing comments)

Commenting code

```
/// <summary>
/// Class Graph represents an immutable graph.
/// . . .
/// </summary>
public class Graph : IGraph
{
   /// <summary>
   /// Computes a string representation of the graph.
   /// The string contains the adjacency lists of each node.
   /// </summary>
   /// <returns>the string representation of a graph</returns>
   public override string ToString()
   {
     StringBuilder myString = new StringBuilder(); // use this to speed up
                                                   // concatenations
     . . .
   }
}
```

Formatting

- Use blank lines to separate groups of code
- Use consistent spacing around operators
- Use indentations
- Align braces (“{“)

[Formatting]

```
#include <stdio.h>

    main(t,_,a)
    char *a;
    {return!0<t?t<3?main(-79,-13,a+main(-87,1-_,main(-86, 0, a+1 )+a)):1,t<_?main(t+1,
    _, a ):3,main ( -94, -27+t, a)&&t == 2 ?_<13 ?main ( 2, _+1, "%s %d %d\n"
    ):9:16:t<0?t<-
    72?main(_,t,"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+\,/+#n
    +,/#;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l
    q#'+d'K#!/\+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n')()#}w'){){nl]'/+#n';d}rw' i;# ){n\l]!/n{n#';
    r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#\n'wk nw' iwk{KK{nl]!/w{%'l##w#' i;
    :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
    ;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;\#'rdq#w! nr'/ ') }+}{rl#'{n' ')# }'+}##(!!/“):t<-
    50?_==*a ?putchar(a[31]):main(-65,_,a+1):main((*a == '/')+t,_,a\+1 ):0<t?main ( 2, 2 ,
    "%s"):*a=='/'||main(0,main(-61,*a, "!ek;dc \i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m
    .vpbks,fxntdCeghiry"),a+1);}
```

Consistency

- recommendation:
  - invent or choose a style, regarding:
    - class names
    - class members
    - constants, local variables
    - spacing
    - alignments . . .
  - stick to it!
    - use it consistently in all your code files.

Published guidelines

- A beautiful list of guidelines for C# code can be downloaded from:

  http://csharpguidelines.codeplex.com/downloads/get/540283

- Third delivery
  - A prototype of your application
    - An archive with the source files for a running version of your software, with more or less full functionality
      - The solution/project/workspace folders/files will be provided
      - No compiled objects (.class, .dll, .obj, .exe, etc.), but
      - *I should be able to compile your sources*.
  - A description/documentation of classes
    - as a separate archive containing
      - document (Word, PDF) or
      - HTML page, or
      - MS Help file.

- Deadline: Friday June 22.

C# lecture

Class libraries

- Class Libraries → DLL files (Dynamic Link Libraries)
  - Help to organize things by grouping related classes and interfaces
  - not executable (cannot be started as programs/applications)
    - their content is used by other libraries or executables
  - easily created and used in .NET, with Visual Studio

both executable programs and class libraries created in .NET are "assemblies"

- they are described by some specific information (name, version, company, etc.)

APIs

- “Application Programming Interface”
  - The set of classes and/or their public methods that are offered by an application or library.

Documenting code

- Similar to C++
  - /* … */ for comments that span more lines
  - // for comments that go to the end of current line
- Special comments: ///
  - contain text enclosed in XML tags (<summary>)
  - VisualStudio code editor generates automatically tags for relevant information (method parameters, return values)

Documenting code

- *check the documentation in Help to see the most important XML tags*

- these special comments can be exported as an XML file
  - in Visual Studio, check “XML documentation file” in the “Properties” dialog of the project (page “Build”)

Documenting code

- use e.g. Sandcastle or Doxygen to generate the final documentation (in MS Help format, or HTML).

Homework

- pinpoint possible problems and style inconsistencies in the following C# code snippet:

```
1.  List<bool> restricted;
2.  List<string> Liste_Bedingung;
3.  SocketComm m_socket;
4.  private bool SetConstraint(int _ndx){
5.          if (restricted[_ndx]) Liste_Bedingung[_ndx] = "OK";
6.          else
7.                  Liste_Bedingung[_ndx] = "nicht erfüllt";
8.          return m_socket.communicate(_ndx + ": " + Liste_Bedingung[_ndx]);
9.  }
10. public bool SetConstraints(int nrConstraints)
11. {
12.         bool ret_val = true;
13.         for (int i = 0; i < 8; i++)
14.         {
15.             bool b = SetConstraint(i);
16.             ret_val = ret_val && b;
17.         }
18.         return ret_val;
19. }
```