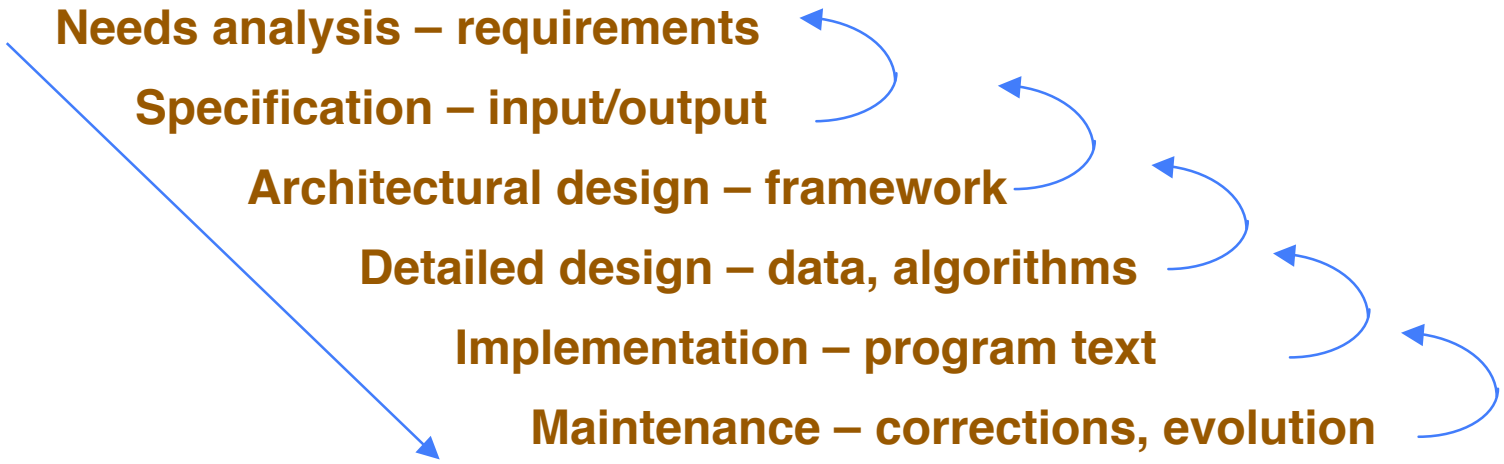
Design
Context and Principles

Waterfall Model – Software Life Cycle

Apply recursively at all levels – from system level to subprogram.
Spiral model and evolutionary development are variations



- At all stages the artifacts produced are documents
  - They may be formal – use mathematics and programming languages
  - They may be informal – use natural language
- At all times strive for correctness and precision

What is Programming?

- Specifying **what** to to do and **when** to do it
- The **what** consists of the following
  - » **At the assembler level the hardwired instructions**
    - > **add, load, store, move, etc.**
  - » **At the Eiffel, C, Java level**
    - > **assignment, arithmetic, read/write**
    - > **Subprogram library, API (Application Program Interface)**
- The **when** consists of specifying in what order to do the "what" operations
  - » **Control structures – these are the only ones**
    - > **sequence** > **choice** > **loop**
- What and when are intertwined – changing one often requires changing the other

What is Design?

- Design is the creation of a plan.
  - » **Consider design as imposing constraints on the "when" and "what" of programming.**
  - » **From this perspective, the entire life cycle is comprised of design at various levels .**
    - > **Design bridges the span from requirements to implementation**
- Design comes from the root to designate, to name
  - » **In design one names objects and their relationships**
  - » **The difficult part is finding the "right" objects and the "right" relationships**
  - » **There must be a correspondence between specification and implementation.**
    - > **The objects and relationships in the specification must correspond to the objects and relationships in the implementation**

Design within the Lifecycle

- Consider the constraints imposed in the software lifecycle
  - » **Putting together a requirements document constrains what can be done from all possible programs to the set of programs corresponding to the requirements**
  - » **The specification formalizes the requirements and in the process adds more constraints.**
  - » **Architectural design adds constraints, and so on.**
  - » **Even implementation (programming) adds constraints by specifying in detail every when and what and so is a part of the design process.**
- At each stage, there are fewer choices for what and when.
- At each stage the choices must be made within the constraints imposed by the earlier choices – or else backtracking to earlier stages is required

Seamlessness

Since design pervades the entire software lifecycle it is important that supporting methods should apply to the entire lifecycle, in a way that minimizes the gaps between successive activities

Corollary: Should be easy to move information among different notations

- formal – program text and mathematics
  - <—> informal – documentation text
  - <—> informal – diagrams

Design for Software Quality – 1

- Readable and understandable
  - » **All Design artifacts – program text included – are primarily to be read and used by people.**
  - » **Execution is incidental**

> **Primary purpose of design is to communicate with other people – even you are somebody else in the future, so you must communicate with yourself**

- Works
  - » **Complete – Correct – Usable**
  - » **Efficient as it needs to be**
    - > **Speed up where necessary after instrumentation**

Design for Software Quality – 2

- Modifiable
  - » **All programs evolve over time**
  - » **Make plausible modifications easy**
    - > **One sign of a good design is it is easy to modify and adapt to changing circumstances**

- On Time and on Budget
  - » **Time is money – pay back on investment**
  - » **Imbedded systems – programs are only a part of the system**

Principles of Public Design

- Principle of Use
  - » **Programs will be used by people**
- Principle of Misuse
  - » **Programs will be misused by people**
- Principle of evolution
  - » **Programs will be changed by people**
- Principle of migration
  - » **Programs will be moved to new environments by people**

High Level Design Goals

- Correctness
  - » **The ability of a software system to perform according to specification, in cases defined by the specification**
    - > **First write correct programs, then worry about efficiency!!!**
    - > **A fast program that is wrong is worse than useless**
- Efficiency
  - » **Use an appropriate amount of resources for the task**
    - > **Space for storing data and temporary results**
    - > **Execution time**
    - > **Space – time tradeoff**
    - > **Communications bandwidth**
- Ease of use – including installation

Implementation Goals – 1

- Robustness
  - » **The ability of a software system to react in a reasonable manner to cases not covered by the specification**
    - > **Works correctly for defined inputs**
    - > **Recover gracefully from unexpected inputs**
    - > **Recover gracefully from hardware and algorithm errors**

- Adaptability
  - » **Modifiable**
  - » **Use in unexpected ways**

Implementation Goals – 2

- Reuse
  - » **The ability of a software system to react in a reasonable manner when reused**
  - » **Use variations in different software products**
    - > **same as ... except ...**
  - » **NOT just using**
    - > **A pot is not reused when boiling water. It is meant to boil water on many different occasions**
    - > **Reuse -- pot is used to bail a boat, maybe by bending it to fit the shape of the hull**

Structural Design Aspects

- Tokenization
  - » **What kinds of symbols are in the input and output**
- Data structures
  - » **How and what data structures should be selected**
- Program structures
  - » **How should a program be structured**
- Procedure partitioning
  - » **How should one decide when a set of operations be made into a procedure**
- Class partitioning
  - » **How to decide what goes into a class or module**
- Correspondence
  - » **When do structures correspond**
  - » **When to use communicating sequential processes**

OO Design Principles

- Abstraction
  - » **Extract fundamental parts**
    - > **Describe what is wanted**
  - » **Ignore the inessential**
    - > **Do not describe how to do it**
- Encapsulation – Information Hiding
  - » **Expose only what the user needs to know**
    - > **The interface**
  - » **Hide implementation details**
- Modularity
  - » **Handle complexity using divide and conquer**
  - » **Minimize interaction between parts**

OO Design Techniques – 1

- Classes and Objects
  - » **Classes define abstract data types**
  - » **Objects are instances of those types**
- Interfaces and Strong Typing
  - » **Interface gives the user what they need to know to use objects from a given class**
    - > **API – Application Program Interface**
  - » **Strong typing compiler enforces objects are used correctly by type**
    - > **Do not take square root of a colour**
- Inheritance and Polymorphism
  - » **Inheritance – single and multiple – provides for reuse**
  - » **Polymorphism invoke the proper method for an object depending upon its type**

OO Design Techniques – 2

- Assertions
  - » **Equip a class and its features with pre and post conditions, and invariants**
  - » **Use tools to produce documentation out of these assertions**
  - » **Optionally monitor them at run time**
- Information hiding
  - » **Specify what features are available to all clients, some clients or no clients**
- Exception handling
  - » **Support robustness with a mechanism too recover from unexpected abnormal situations**

OO Design Techniques – 3

- Genericity
  - » **Write classes with formal generic parameters representing arbitrary types**
- Constrained genericity
  - » **Combination arising from genericity and inheritance to constrain formal generic parameters to a specific type**
- Redefinition of features and deferred features
  - » **Reuse requires the ability to modify an object for a new environment so features can be redefined**
  - » **Some design decisions must be deferred so provide a means to specify the interface of a feature without defining how it does it.**