**Abstractions**

**Abstractions**

* Managing a bunch of different objects can be confusing when all the implementation details must be remembered
* *Abstraction:* separating the purpose of an object from its implementation by hiding unnecessary details
  + *Allows the use of an object without worrying about how it works*
  + *Allows large problems to be managed easier*
    - *Multiple modules can be combined to form large programs without dealing with minute details*
    - *Results in more efficient code which is easier to write, easier to use, and easier to maintain*

**Data Abstraction**

* Applying abstraction to data allows us to create data types which better match the data we wish to store, while also hiding cumbersome details about the underlying data structure
* *Data Structure:* an implementation dependent collection of data and relationships between data elements
  + *Example:* arrays, etc.
* *Abstract Data Types:* an implementation independent collection of data with unique properties and a complete set of operations on the data. A generalized description of a concrete data type (e.g., int, char, boolean, etc.). It specifies the operations that can be performed on objects of that type

**Abstract Data Types (ADT)**

* Provides a clearly defined *interface* to a collection of data
  + *Interface:* the way data is interacting between different data areas
* The user is only concerned with the interface
  + Only the creator of the ADT worries about the implementation
* Categories:
  + Linear, Hierarchical, Graphs, Unordered
* Common Operations:
  + Search, Retrieval, Removal, Insertion, Traversal, etc.

**Example ADT’s**

* Bag and Set
  + Both contain items. A Bag can have duplicate items, a Set cannot
* Date
  + getDay, getMonth, getYear
  + setDay, setMonth, setYear
* String
  + Contains
  + Length
  + Concat
  + Substring

**Software Development and Engineering**

* Software Engineering focuses on applying a formal framework to software development
* The goal is to improve software development
  + Reliability
  + Efficiency
  + Maintainability
  + Reuse
* ADTs provide a set of tools that can be used to simplify the Software Development Process resulting in better software

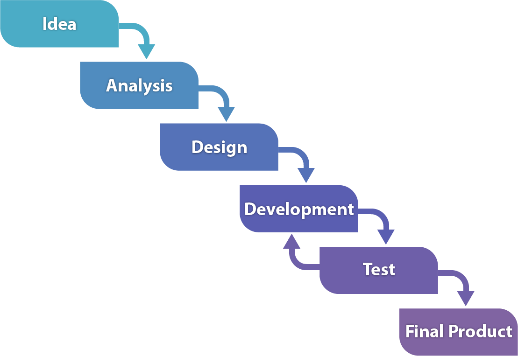
**Software Process Models**

* Build-and-Fix Model

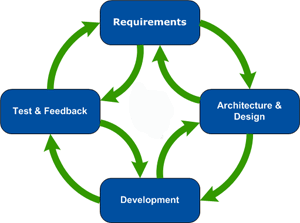
Modify until customer is satisfied

Implement First Build

* Model from Tymann text (Waterfall Model)



* Waterwheel Model



**Development Phases**

* Request
  + Client submits broad requests for software to solve a given problem
* Analysis
  + Software Engineer(s) creates formal specifications for exactly what the program will do
  + Verify correctness of specifications
* Program design
  + Develop formal plan for how the program will work and how it will be implemented
  + Verify correctness of design
    - Formal: design inspection
    - Informal: design walk-through
* Implementation
  + Code the project in the appropriate language(s)
  + Test to insure correctness
* Integration
  + Multiple modules must be combined together or with existing systems
  + Test to insure parts work correctly together
* Maintenance
  + Software has a long life span (typically 5-15 years)
  + Changes must be made to fix problems or to accommodate changes in requirements or technology

**Software Process Models cont.**

* Iterative Model

**Problem Specification**

* Goal:
  + Creation of a complete, accurate, and unambiguous statement of the exact problem
    - Know exactly what to do

**Program Design**

* Divide and conquer
  + modularize the problem and solve one part at a time integrating all the parts to develop a solution
* Top down
  + An overview of the problem is formulated without going into the details of each part
  + Each part is then refined again until the design is complete enough to move to implementation
  + It can be difficult to modify the final program to meet new and unexpected requirements
* Bottom UP
  + Each part is specified in detail and even implementation
  + As each part is developed, the parts are linked together to form a partial solution
  + The partial solutions are integrated to create a complete system
* Object Oriented Design
  + Problem is decomposed into entities
    - Classes model a distinct type of entity
    - A specific instance of an entity (class) is an object
  + This is the approach used in Java programs

**Design Documentation**

* High-Level
  + Specifies the modules of a program and how they interact
  + Useful for medium to large size projects in which many modules exist and must communicate (work together) to solve the problem
  + Unified Modeling Language (UML)
* Detailed level design
  + Specifies the design of each module
  + Level of detailed is enough that the programmer’s job is to translate the design into program code
  + Javadoc

**Testing Code**

* Unit testing
  + Test performed on single class
  + Performed with simple test application that tests all possible behavior of class

**Types of Testing**

* Haphazard
  + Randomly trying obvious input data
  + Easiest, but least effective
* Black-Box
  + Possible inputs divided into clusters, and samples from each cluster are tests
* White-Box
  + Develop teste data that tests all parts of program by examining code
* Code inspections and walk-throughs can also be used to verify code

**Coding Styles**

* Good style makes coding simpler and produces better code – more maintainable and reusable
* The appearance of your work makes a difference
* Elements of Style:
  + Meaningful identifiers
  + Indentation
  + Capitalization
  + Documentation
    - Pre-Condition: condition which must be met before a method is called for it to work correctly
    - Post-Condition: conditions guaranteed to be true when the method finishes execution

**Contemporary Techniques**

* Exception handling
* Assertions
* Streams
* Graphical user interfaces (GUI)
* Concurrent programming (multi-threading)
* Network Programming