CSIS 3103

Introduction to Data Structures

Objectives

- Understanding and using data abstraction
- Designing and using efficient data structures and algorithms
- Applying the techniques of object oriented programming
- Gaining skill with professional grade software development tools
- Applying software engineering techniques

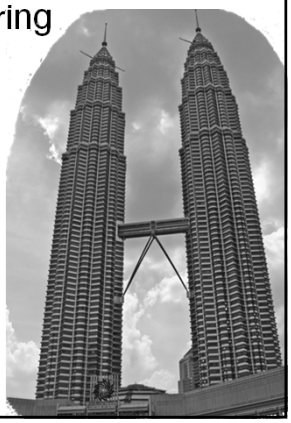
Result: High quality software systems

What is "*quality software*"?

– It solves the right problem
– It works (correct and robust)
– Is modifiable without excessive time & effort
– Is reusable
– Is completed on time and within budget
– Is efficient
– Is well documented
– etc., etc., etc…

Software Engineering

The theory and techniques that underlie the development of high quality software



Why Does it Matter?

Complex data structures and algorithms are used in all significant software systems

– Data compression uses *trees* (MP3, GIF, etc.)
– Networking uses *graphs* (routers and telephone networks)
– Operating systems use *queues* and *stacks* (Scheduling and recursion)
– Security uses complex math algorithms: (GCD and large primes)

Data Structures

- The study of how data is organized, manipulated, and used by computer programs
- Ways to organize large amounts of data

Bits are grouped into strings to form
- integers
- floats
- characters
- character strings, . . .

```
10011101000101101010101110101001010111110000101010001…
```

Abstract Data Types

Domain: The set of values represented by the type
- *attributes*

Operations: Methods for processing values from the domain
- *behavior*

ADT Design & Implementation

Specification
- Describe *what* an ADT does

Application
- Determine *where* the ADT would be useful (kinds of problems it might help solve)

Implementation
- Determine *how* to code it
  - concrete representation of data components
    - different choices may affect use of memory
  - algorithms implement the operations
    - alternative algorithms may depend on data representation
    - can effect execution time

A Design Problem

- You've been hired by a company that provides Internet content infrastructure. They need a data structure to look up URLs by keyword.
- There are many billions of documents on the Web
- The client has decided that ≈100 thousand keywords are enough

```
widget   ⇒   http://images.acme.com/widget.jpg
```

The Questions

- What data structure should you use?
- How long will a lookup take?
- What about adding new keyword/URL pairs?
- Should we be worried about storage requirements as well as lookup time?

Possible Solutions

- **A linked list of keyword/URL pairs**
  linear lookup time
- **An array sorted by keywords, lookup using binary search (divide and conquer)**
  logarithmic lookup time
- **A binary search tree, keyed by keywords**
  logarithmic lookup time, under the right conditions
- **A hash table**
  constant lookup time, under the right conditions
- **Something else?**

And the Problem Will Change

- **Web content doubles roughly every 6 months**
  - Space may be more important that time
  - Will the software we design still work in 5 years?
- **A more intelligent search on content**
  - e.g., “automobile” should also get things with “car”
  - allow multiple keywords in searches
- **Can we design software that will adapt?**
  - More frequent adds/deletes

A One Shot Deal?

Can we use the same solution for items in an Ebay database? Twitter tweets?

Key idea:

What details can be "abstracted" in the software, so they can be reused later?

Abstraction & Information Hiding

Abstraction: Leave out complex details and concentrate on the essentials
- Control abstraction (if, while,…)
- Procedural abstraction (methods)
- Data abstraction (types)

Information Hiding: Hiding the details of a module to control access from the rest of the system

The Software Life Cycle

- Software products go through several stages as they mature from initial concept to finished product
- The sequence of stages is called a *life cycle*
- Design and document software in an organized way so that it can be easily understood and maintained after the initial release
- The person who maintains the software is not necessarily the person who writes it

Software Life Cycle Activities

Certain activities are essential for software development
- Problem definition
- Requirements specification
- Architectural, component, and detailed designs
- Implementation
- Unit, integration, and acceptance tests
- Installation and maintenance

Problem Definition

The *client* proposes a problem

This client could be:

a friend or associate;
another department in your organization;
an external organization;
yourself

The problem must be well-defined.

Requirements Specification

- Produces a precise statement of the problem
- Researches similar problems and solutions
- Determines resources required
- Produces a time frame for the solution

Result: A *requirements document*

- Specifies precisely what is to be produced
- Includes criteria for the testing team
- Used to validate the solution
- The *user manual* will be derived from the requirements document

Design

The *design team* determines:

- an outline of the solution;
- the software objects needed;
- other software components needed;
- which parts already exist:
  - libraries
  - proprietary packages

Implementation

This is where the actual *coding* is done

- Individual components are implemented separately
- All code must be thoroughly documented
- All code must be thoroughly tested

Testing

- Individual components tested (*unit testing*)
  - Watch for special cases (boundary values)
- The entire system is assembled and tested
- Does the solution solves the problem correctly?
  - May result in a revision of the requirements document

Installation & Maintenance

Long-term maintenance includes:

- user support
- patches (“service packs”)
- revisions

Implementation Goals

Robustness: The program shouldn’t crash or do something “unexpected” regardless of the input

Adaptability: The program should be easy to modify in order to solve related but unforeseen problems

Reusability: Software modules or packages should be general enough to use in a wide range of applications