Applied Component-Based Programming for Engineers and Scientists

CSE 668 / ECE 668

Prof. Roger Crawfis

Introduction

- Overview of the course
- Software Engineering history (my perspective)
- Component-based Programming
- Overview of .NET

Course Overview

- Software Engineering Design
- Reusable Components
- Windows .NET platform and C#
- Applied Engineering context
- Hopefully lots of discussions
- Homeworks – small assignments or labs.
- Final Project
- Midterm

Focus is on the practical and pragmatic aspects of software development, rather than theory.

Component-Based Programming

- Programming versus Software Engineering
  - Need to maintain
  - Need to extend
  - Avoid the "not-written here" or "re-inventing the wheel" syndrome.
- Decompose into reusable components
- First, some history

Procedural Software Development

- Scientific computing has relied on procedural programming concepts
  - Old Fortran decks.
- Functional decomposition and Structural Analysis
  - Yourdon design – Dataflow diagrams, data dictionaries,
  - Waterfall method
- Reuse was in general libraries
  - Even scientific computing did not embrace this, due to the difficulty of interfacing the library with the data.
- Late 1970's to present
- I loved it!!

Procedural Programming

- During this time many software metrics were introduced:
  - Lines of code
  - Cyclomatic or conditional complexity
    - Number of possible paths through a section of code.
  - **Cohesion**
    - How focused is the method (want High Cohesion)
  - **Coupling**
    - Low, loose or weak coupling is desired.
    - High, tight or strong coupling is undesired.
    - Measures the relationships between various methods.
  - Coupling and Cohesion usually go hand in hand. High cohesion leads to low coupling and vice versa.

The Waterfall Method

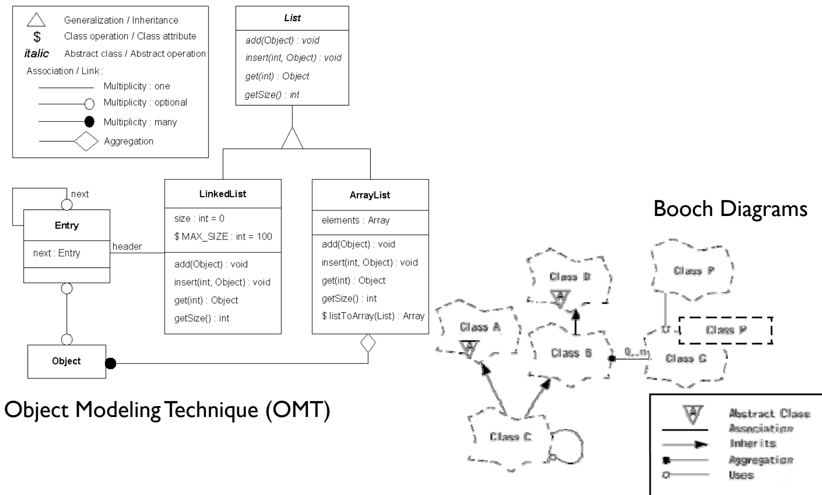
- Another good result from the 1970's and 1980's was the development of processes for software development. Many books on the subject, gov't mandates, etc.
- Peopleware
  - *coined by DeMarco and Timothy Lister in the popular 1987 book "Peopleware : Productive Projects and Teams".*

"The major problems of our work are not so much technological as sociological in nature."
– DeMarco and Lister

Object-Oriented Software Development

- Started with new languages like Smalltalk (but influenced by Simula67).
- Took off with Ada, C++, Eiffel.
  - The US Department of Defense at one point in time mandated that all new government contracted software had to be done in Ada.
- Java followed once the web took off.
- Booch diagrams
- Objects package the data with the behavior that modifies or uses that data together.

Object-Oriented Software Development



Object-Oriented Reuse

- Classes in C++ allowed for reuse within that system.
  - Reuse requires the source code (header files).
  - Reuse also through inheritance
- Some class libraries started to appear (usually from a commercial vendor), allowing reuse across multiple projects.
- The open-source movement (sourceforge.net, etc.) has made many (too many) class libraries available.
  - No standardization.
- Took awhile before people accepted non-home grown packages (like the Standard Template Library).
- Sun provided many reusable structures in Java.
  - Continues to grow.

Software Engineering and OO

- Simply using object technology, and a language which supports object orientation does not automatically lead to systems, which are robust and stable under changes.
- Old functional programming concepts die hard.
  - I built a texture system for a flight simulator with 2-4 classes.
- The concepts of Cohesion and Coupling still apply, as do many of the other metrics.
- Inheritance is inherently highly coupled.
- What is a class?
  - Data structure?
  - Behavior specification – the "is a" relationship?

Design Patterns

- Besides the reuse of actual code, there is the reuse of *solutions to common problems* (aka knowledge).
- Design Patterns were made popular by the book Design Patterns from Gamma, et al.
  - Published knowledge
  - Vocabulary
  - An initial 23 patterns
- Note: Design patterns **add** complexity. If you do not have the problem, why solve it?
- Most of the patterns are centered around either de-coupling two systems or making your system more flexible (and hence more reusable).

Modern Software Engineering

- Program to interfaces
  - Reduce Coupling
- Design Patterns
- Iterative and Incremental development
  - Agile methods, Rapid Prototyping, etc.
- Use case driven processes
- Test driven development
  - Extreme Programming
- Model driven development
- Unified Process Framework
  - Rational / IBM's RUP
  - OpenUP

Java and C#

- Next generation of languages
- More than just a language
  - Virtual machine
  - Comprehensive libraries
  - Security-centric
- The Java Platform
  - You can not divorce Java from the JVM
  - Much more powerful with Networking, Collections, Swing, …
- The .NET Platform
  - You can not divorce C# from the CLI.
  - Much more powerful with Networking, Collections, Forms, …

Java and C#

- With these systems, we can finally move away from system issues and focus on application issues.
  - Unfortunately, not entirely true, but it is improving.
- Application development
  - Configure several robust ***components*** together to build a larger system.

What is a Component?

- Open to wide interpretation. Being misused as the new golden child of the IT acronym / terminology soup.
- Key characteristics of a component:
  - Higher-level of abstraction than objects (aka not a Customer)
  - Communication between components is usually via messages
  - Non-context specific (the state of another component does not influence it unless that component is passed into it)
  - Encapsulated – you can not determine its implementation
  - A unit of independent deployment and versioning
  - Composable with other components
  - Written to a well-defined interface or contract
  - Often used in distributed systems

What is a Component?

- It's relative, as Crocodile Dundee said:
  - "That's not a knife, now that's a knife"
- You will get a better feel for it as we progress through the quarter.
- Basically, when we develop parts of our system (an interface, class or collection of interfaces and classes), do not ask how you would use this, ask how would a Visual Basic application on a remote machine use this? Without access to my source code!

Component Technologies

- Corba - Common Object Requesting Broker Architecture
  - Map the IDL language to your native language
    - Very ugly
- Microsoft's COM
  - Binary Type System
  - Metadata
  - Interface or contract based – the unit of reuse.
    - Resort to IUnknown interface
    - Ugly
  - Runtime-loading – allows copy and paste of objects.
- Enterprise JavaBeans
- .NET Remoting

Component Plumbing

- Component Technologies attempt to codify and ease the necessary plumbing issues inherent in distributed computing:
  - Security
  - Instance Management
  - Type persistence and marshalling
  - Synchronization across machines
  - Hosting
  - …

Path Forward

- We will examine the .NET Remoting and possibly the Windows Communication Foundation support for Components.
- Our focus will be on C#, but any .NET language can be used.
- There is a current buzz about Service-Oriented Programming, Service-Oriented Architectures (SOA) and Application Services.
  - Next course (I think)

Homework

- Read IEEE Software Best Practices articles (real short):
  - ***Keep It Simple***, Steve McConnell (1996) http://www.stevemcconnell.com/ieeesoftware/bp06.htm
  - ***Missing in Action: Information Hiding***, Steve McConnell http://www.stevemcconnell.com/ieeesoftware/bp02.htm
- Read Chapter 1 (14 pages) and start reading Chapter 2 in ***Programming .NET Components***, by Juval Löwy

Resources

- Books
  - Design Patterns – The Head-First series is better than Gamma, et al, especially for journeyman
  - Code Complete 2 – Very detailed advice. Required reading in some companies
    - The Construx site (http://www.construx.com) has some nice resources
  - Refactoring – Refactoring : Improving the Design of Existing Code by Martin Fowler
    - Introduces the concept of code smells
  - Effective … series (e.g., Effective C++, Effective C#, Effective Java).
  - UML – ***UML Distilled*** by Fowler is nice and concise.