Design Hierarchy

The development process culminates in the creation of a system.

First we describe the system in terms of components, and describe those components in terms of sub-components, and describe those . . .

This process requires applying the concept of abstraction, hiding details of components that are irrelevant to the current design phase.

The process of component identification is top-down, decomposing the system into successively smaller, less complex components.

This must be followed by a process of integration, which is bottom-up, building the target system by combining small components in useful ways.

©2000-2002 McQuain WD & Keller BJ

Procedural Paradigm

Apply procedural decomposition: divide the problem into a sequence of simpler sub-problems to be solved independently.

The resulting program will consist of a sequence of procedure calls.

The designer thinks in terms of tasks and sub-tasks, identifying what must be done to whom.

The key idea is to identify simple, constrained tasks which are easy to implement.

Traditional procedural languages: COBOL, FORTRAN, Pascal, C

Design notations: structure charts, dataflow diagrams

©2000-2002 McQuain WD & Keller BJ

Problems in Procedural Development

The result is a large program consisting of many small procedures.

There is no natural hierarchy organizing those procedures.

It is often not clear which procedure does what to what data.

Control of which procedures potentially have access to what data is poor.

These combine to make it difficult to fix bugs, modify and maintain the system.

The natural interdependence of procedures due to data passing (or the use of global data, which is worse) makes it difficult to reuse most procedures in other systems.

©2000-2002 McQuain WD & Keller BJ

Some Software Engineering Goals

Reusability

- develop components that can be reused in many systems
- portable and independent
- "plug-and-play" programming (libraries)

Extensibility

- support for external plug-ins (e.g., Photoshop)

Flexibility

- design so that change will be easy when data/features are added
- design so that modifications are less likely to break the system
- localize effect of changes

©2000-2002 McQuain WD & Keller BJ

Modular Programming

This is a relatively simple extension of the purely procedural approach.

Data and related procedures are collected in some construct, call it a module.

The module provides some support for hiding its contents.

In particular, data can only be modified by procedures in the same module.

The design process now emphasizes data over procedures. First identify the data elements that will be necessary and then wrap them in modules.

Typical languages: Ada 83, Modula

©2000-2002 McQuain WD & Keller BJ

Problems with Modular Programming

Modules do solve most of the (noted) difficulties with procedural programming.

Modules do allow information hiding.

But, you only have encapsulation if data of type is stored in a module.

It is more natural (we claim) to think of the data as the fundamental thing.

There are copy issues. . .

©2000-2002 McQuain WD & Keller BJ

Object-Oriented Paradigm

Think of building the system from parts, similar to constructing a machine.

Each part is an object which has its own attributes and capabilities and interacts with other parts to solve the problem.

Identify classes of objects that can be reused.

Think in terms of objects and their interactions.

At a high level, think of an object as a thing-in-its-own-right, not of the internal structure needed to make the object work.

Typical languages: Smalltalk, C++, Java, Eiffel

©2000-2002 McQuain WD & Keller BJ

Why Object-Oriented?

First of all, OO is just another paradigm.

Any system that can be designed and implemented using OO can also be designed and implemented in a purely procedural manner.

But, OO makes some things easier.

During high-level design, it is often more natural to think of the problem to be solved in terms of a collection of interacting things (objects) rather than in terms of data and procedures.

OO often makes it easier to understand and forcibly control data access.

Objects promote reusability.

©2000-2002 McQuain WD & Keller BJ

Dirty Secrets

Objects must interact.

Those interactions are achieved via object methods, which are just procedures.

Procedural decomposition still applies, but the procedures themselves are more tightly bound to the data.

The system (in operation) is still just a sequence of procedure calls.

But. . . we think of the system as a collection of interacting objects.

©2000-2002 McQuain WD & Keller BJ