Analysis

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.

Sobering Truths

Is design important? 75%-80% of system errors are created in the analysis and design phases.

Analysis and design phases account for about only 10% of the overall system cost. (Perhaps you get what you pay for.)

Only about 25% of software projects result in working systems.

Some SE 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., Eclipse, 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

OO Analysis

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

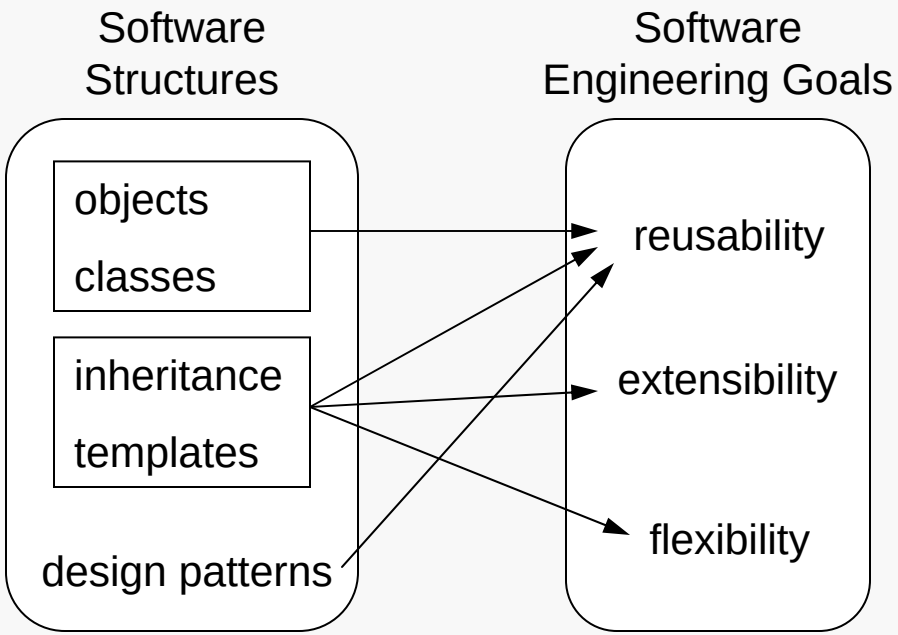
OO and SE Goals

Objects and classes help programmers achieve a primary software-engineering goal: reusability

A single class is used repeatedly to create multiple object instances.

More importantly, encapsulation prevents other developers from inadvertently modifying an object's data.

Separation allows different implementations to be used for an interface.



Identifying Objects and Classes

We must:

- identify potential objects from the specification
- eliminate phony candidates
- determine how the legitimate objects will interact
- extrapolate classes from the objects

This process:

- requires experience to do really well
- requires guidelines, none of which is entirely adequate
- often uses several approaches together
- should lead to too many rather than too few potential objects

Identifying Objects and Classes

Abbott and Booch suggest:

- use nouns, pronouns, noun phrases to identify objects and classes
- singular → object, plural → class
- not all nouns are really going to relate to objects

Coad and Yourdon suggest:

- identify individual or group "things" in the system/problem

Ross suggests common object categories:

- people
- places
- things
- organizations
- concepts
- events

Some Categories of Classes

tangible things from the problem domain

system interfaces and devices

agents, providing objects to carry out operations

events and transactions, something done

users and roles

sub-systems

external systems

containers

Analysis of the GIS Specification

A little bit of Abbot and Booch yields some candidate objects/classes:

- file
- GIS record
- geographic coordinate
- bucket PR quadtree
- index entries
- FID
- BST
- file offset

More Analysis of the GIS Specification

A logical consideration of the specified system suggest some more:

- parser for the command file
- parser for the GIS file
- FID index
- geographic coordinate index
- controller
- system initializer
- latitude and longitude

Classification and Elimination

Some "objects" do not require the design of a new class:

- file — language classes will do for this
- file offset — just a primitive/language type
- latitude and longitude — Do these need to be a new type?
  How will we actually model them?
- geographic coordinate — Is this needed? Could be a base type for lat/long, but…?

Objects for Abstraction/Localization

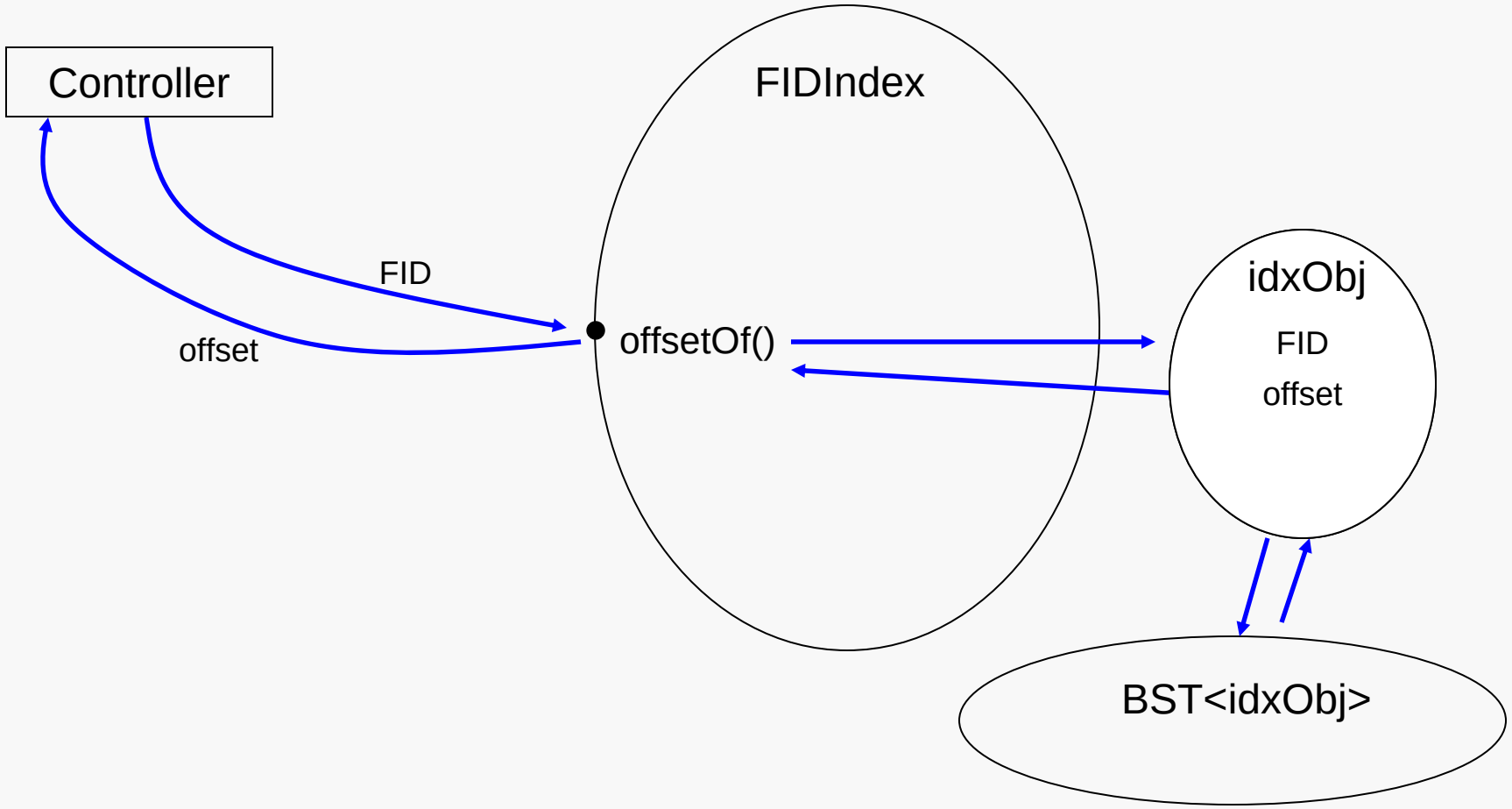
Some candidates provide abstract interfaces that localize constraints:

- parser for the command file — encapsulates the effect of command file formatting
- parser for the GIS file — does the same for the GIS record files

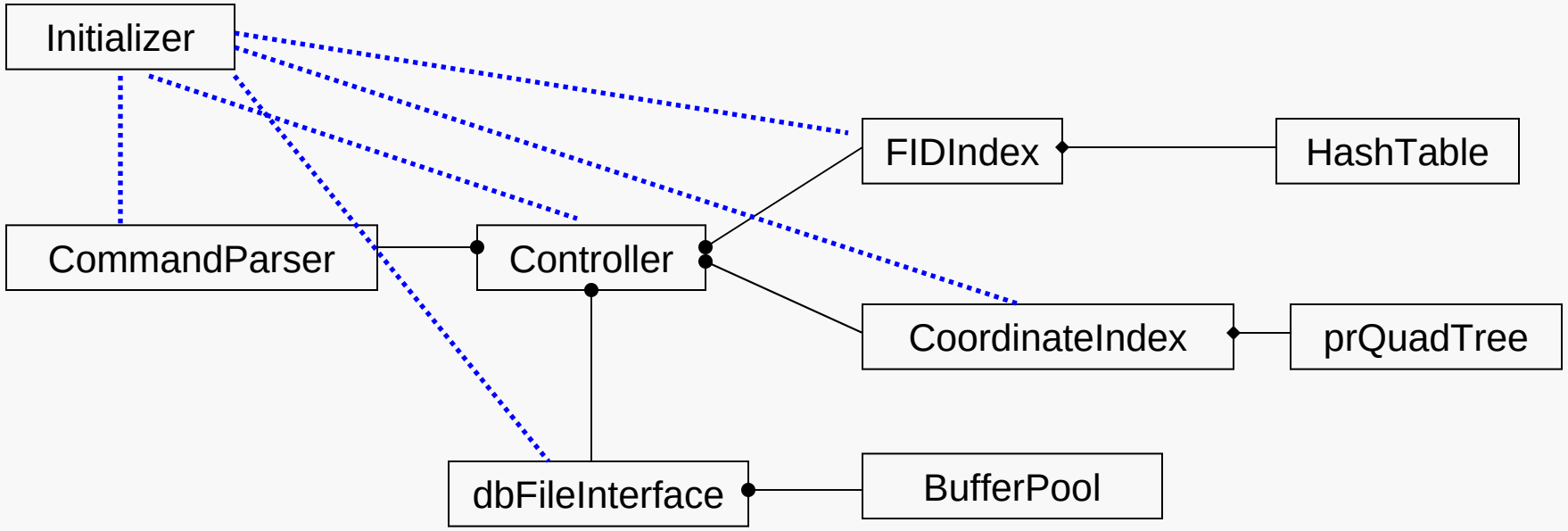
These will make it easier to deal with changes to external file specifications.

Objects to Define Interfaces

Some candidates provide more appropriate interfaces for client transactions:



Possible System Organization



There would also be some other classes/objects, including:

- index objects used only by the `FIDIndex` and `BST`
- index objects used only by the `CoordinateIndex` and `prQuadTree`