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.

©2000-2010 McQuain

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.

Only about 25% of software projects result in working systems.
(Perhaps you get what you pay for.)

©2000-2010 McQuain

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

©2000-2010 McQuain

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

©2000-2010 McQuain

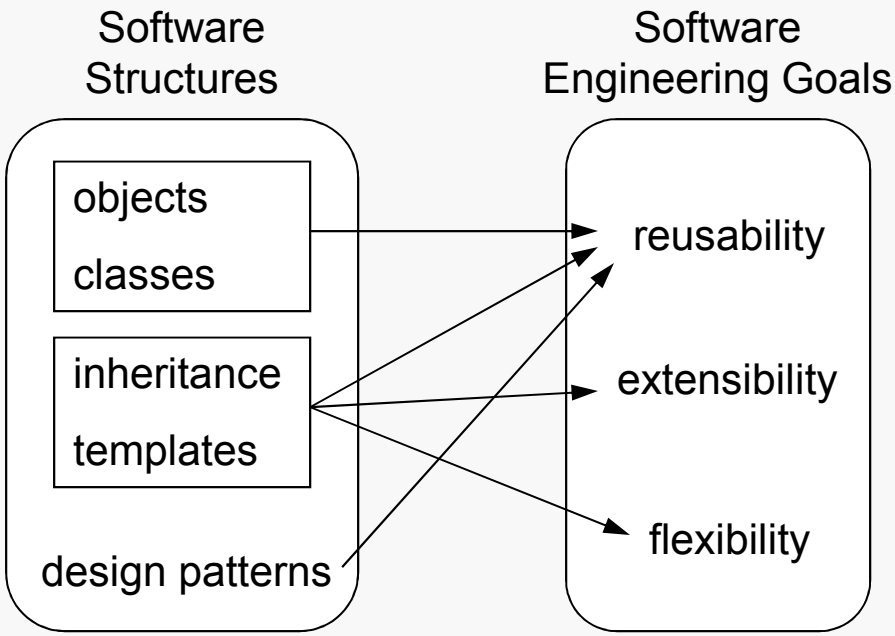
OO and SE Goals

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.



©2000-2010 McQuain

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

©2000-2010 McQuain

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

©2000-2010 McQuain

Analysis: FileNavigation Specification

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

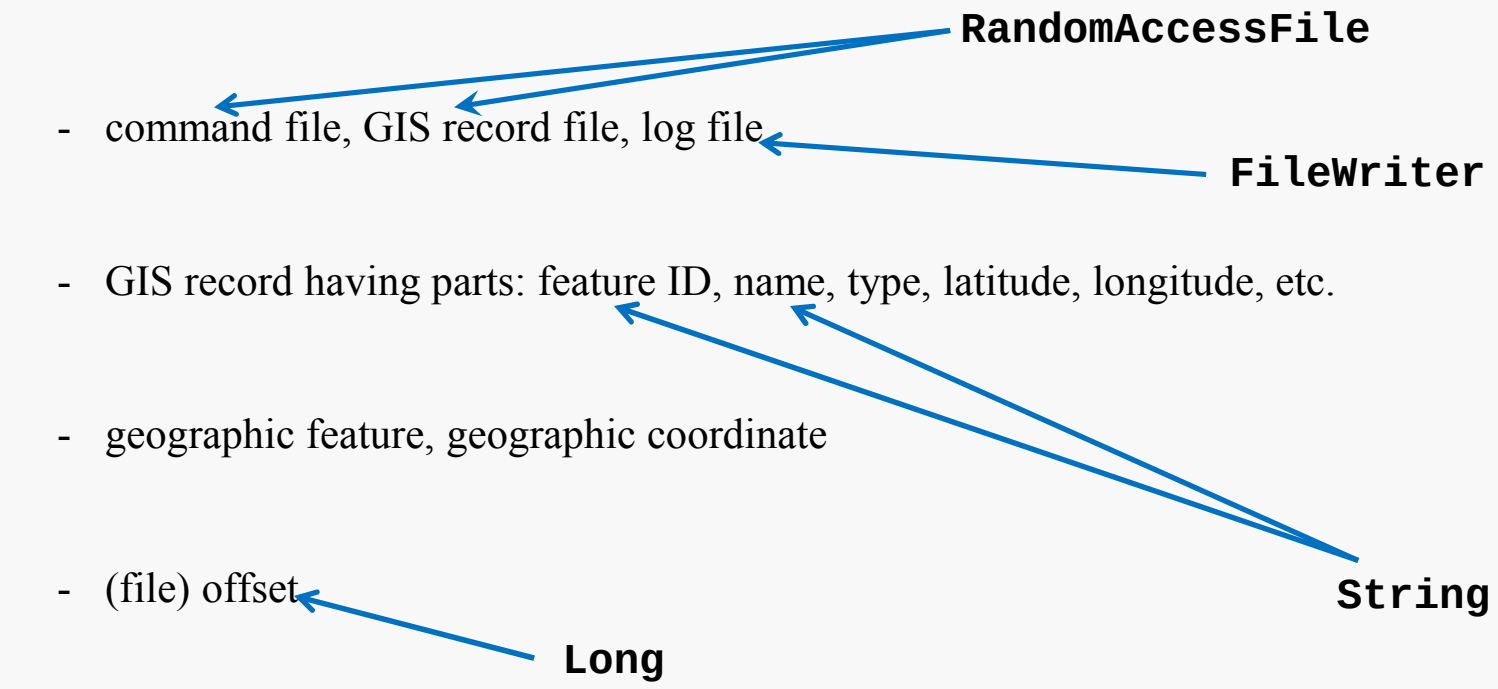
- command file, GIS record file, log file
- GIS record having parts: feature ID, name, type, latitude, longitude, etc.
- geographic feature, geographic coordinate
- (file) offset
- command

There are other nouns and noun phrases that could be considered.

©2000-2010 McQuain

Analysis: Map Candidates to Language

Some of these candidates appear to just map to Java language types:



- command file, GIS record file, log file
- GIS record having parts: feature ID, name, type, latitude, longitude, etc.
- geographic feature, geographic coordinate
- (file) offset
- command

©2000-2010 McQuain

Analysis: Map Candidates to Language

Other candidates should be user-defined types:

`class GISRecord`

- GIS record having parts: feature ID, name, type, latitude, longitude, etc.

`class Longitude`

`class Latitude`

- geographic feature, geographic coordinate

`class GeoCoordinate`

- command

`class Command`

Do these make sense logically?

Do they play an important role in the system?

Do they promote flexibility and reusability?

©2000-2010 McQuain

Analysis: FileNavigation Project

Reconsidering the specification and looking for roles that need to be played and implied elements (Coad & Yourdon):

- command file parser, GIS record file parser (*interface abstractions*)
- command processor (*event loop handler*)
- controller (*organizer and driver*)

Again, do these make sense within the context of the system?

Do they play important roles?

Do they promote flexibility and reusability?

©2000-2010 McQuain

Objects for Abstraction/Localization

Some candidates provide abstract interfaces that localize constraints:

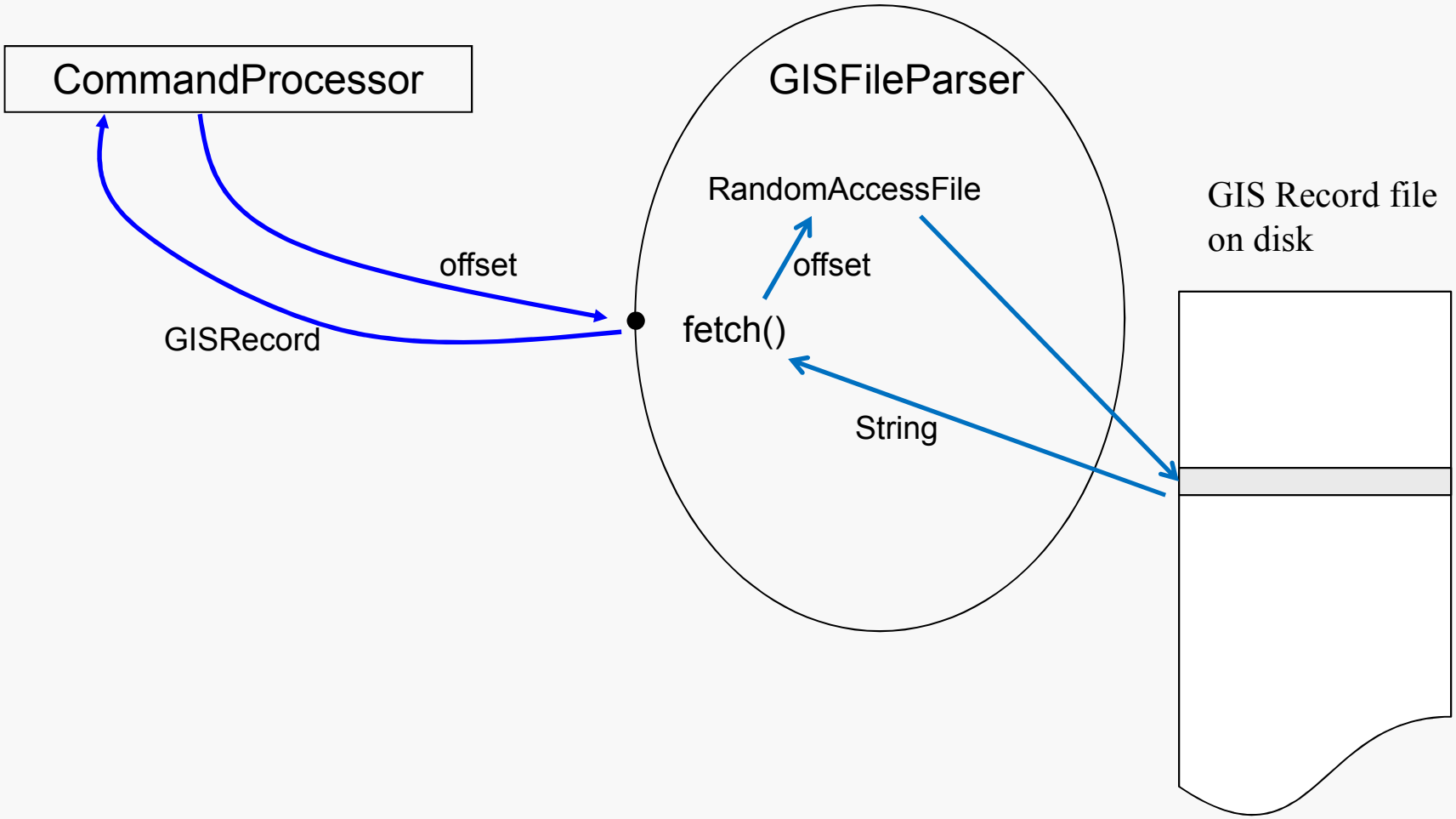
- parser for the command file — encapsulates the effect of command file formatting
- parser for the GIS file — does the same for the GIS record files

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

©2000-2010 McQuain

Objects for Abstraction/Localization

The effect of using a "front-end" for the data file:



©2000-2010 McQuain