OO Frameworks

Jonathan I. Maletic, Ph.D.
<SDML>
Department of Computer Science
Kent State University

Introduction

- Frameworks support reuse of detailed designs and architectures
- An integrated set of components
- Components collaborate to provide a reusable architecture for a family of related applications

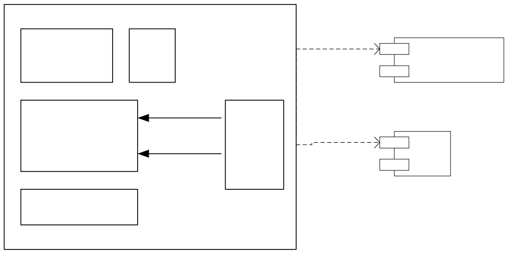
Using Frameworks

- Frameworks are semi-complete software applications
- Complete applications are developed by
  - Inheriting from and
  - Instantiating parameterized framework components
- Frameworks provide domain specific functionality
  - Business, telecom, databases, OS, etc.
- The framework determines which objects and methods to invoke in response to events

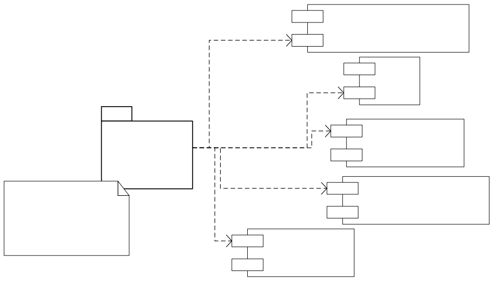
Frameworks vs. Libraries vs. Patterns

- Frameworks
  - – Reusable semi-complete application
  - – Main body and algorithm
- Class library
  - – Self contained
  - – Pluggable ADTs
- Patterns
  - – Problem, solution, context

Framework Architecture

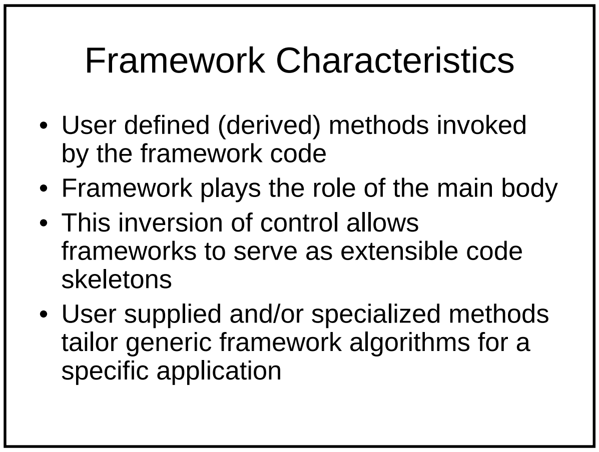


Library Architecture



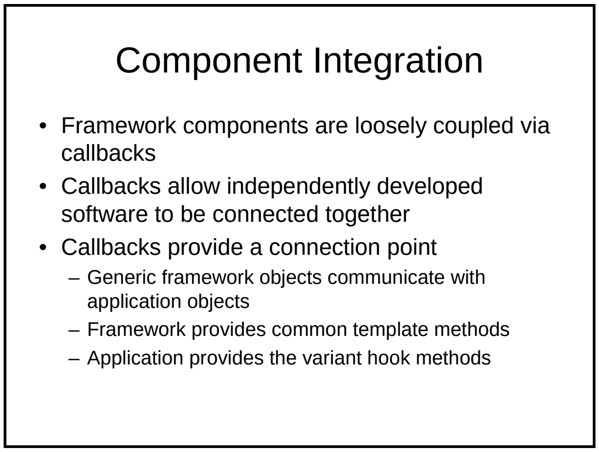
Networking UI

Math Libaray



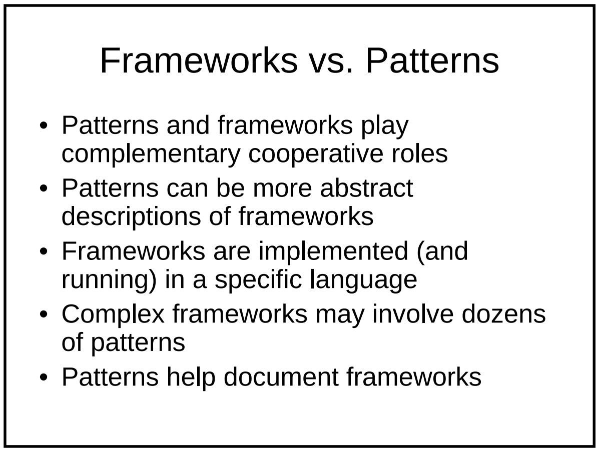
Framework Characteristics

- User defined (derived) methods invoked by the framework code
- Framework plays the role of the main body
- This inversion of control allows frameworks to serve as extensible code skeletons
- User supplied and/or specialized methods tailor generic framework algorithms for a specific application



Component Integration

- Framework components are loosely coupled via callbacks
- Callbacks allow independently developed software to be connected together
- Callbacks provide a connection point
  - Generic framework objects communicate with application objects
  - Framework provides common template methods
  - Application provides the variant hook methods



Frameworks vs. Patterns

- Patterns and frameworks play complementary cooperative roles
- Patterns can be more abstract descriptions of frameworks
- Frameworks are implemented (and running) in a specific language
- Complex frameworks may involve dozens of patterns
- Patterns help document frameworks

Object Oriented Frameworks

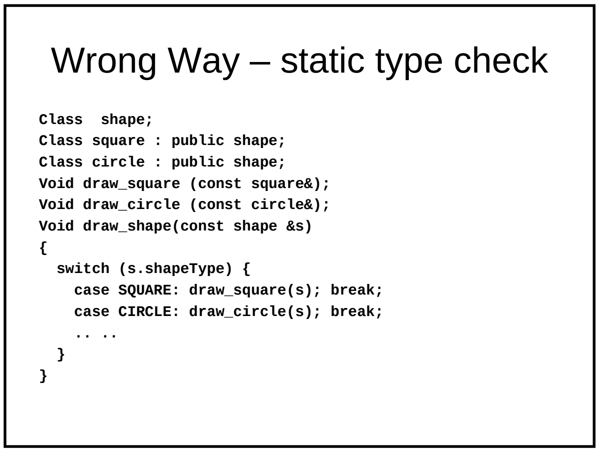
- Aka Object oriented abstract design
- Consists of:
  - Abstract class for each major component
  - Interfaces between components defined in terms of sets of messages
  - Normally a library of subclasses that can be used as components in the design
- Examples:
  - Modern UI toolkits – JavaAWT, MFC
  - HippoDraw

Open vs. Closed

- Determining common and variable components is important
  - Insufficient variation makes it difficult for users to customize framework components
  - Insufficient commonality makes it hard for users to understand and depend upon framework behavior
- Generally, dependency should always be in the direction of stability
  - Components should not depend on any component less stable than itself
- Open/Closed Principle:
  - Allows most stable components to be extensible

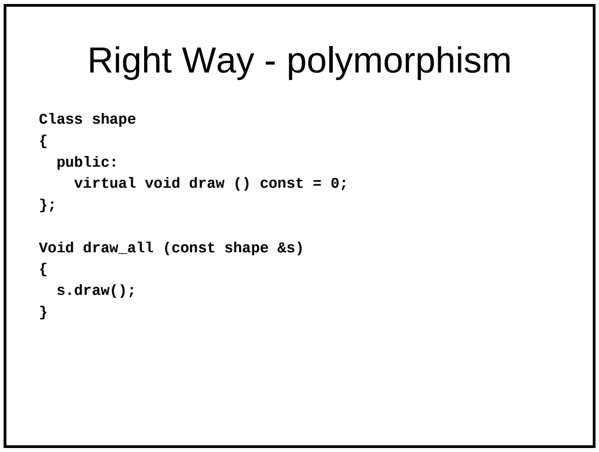
Open/Closed Principle

- Components should be:
  - Open for extension
  - Closed for modification
- Implications:
  - Abstractions is good
  - Inheritance and polymorphism are good
  - Public/global data is bad
  - Runtime type identification can be bad



Wrong Way – static type check

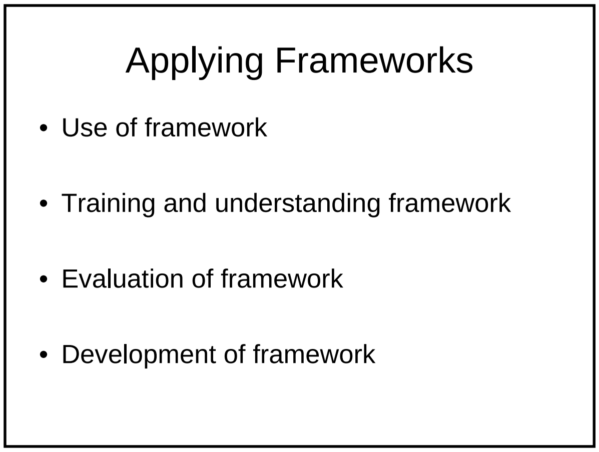
```
Class  shape;
Class square : public shape;
Class circle : public shape;
Void draw_square (const square&);
Void draw_circle (const circle&);
Void draw_shape(const shape &s)
{
  switch (s.shapeType) {
    case SQUARE: draw_square(s); break;
    case CIRCLE: draw_circle(s); break;
    .. ..
  }
}
```



Right Way - polymorphism

```
Class shape
{
  public:
    virtual void draw () const = 0;
};

Void draw_all (const shape &s)
{
  s.draw();
}
```



Applying Frameworks

- Use of framework
- Training and understanding framework
- Evaluation of framework
- Development of framework

Building Applications

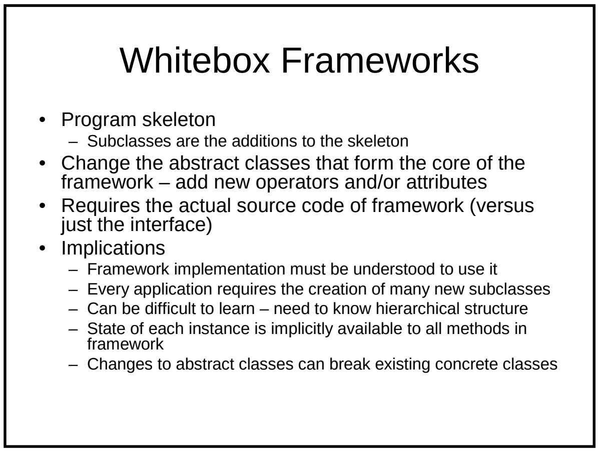
- An application developed using a framework includes:
  - Framework
  - Concrete subclasses
  - Scripts that specified which concrete classes to use and how to interconnect them
  - Objects that have no relationship to framework (utilities and domain specific)

Blackbox Frameworks

- Customize framework by supplying it with a set of components that provide application specific behavior (e.g., GUI frameworks)
- Connect existing components
- Does not require changes to framework and no new concrete subclasses
- Reuses framework's interface and rules
- Analogous to building from legos and connecting ICs
- Application programmers only need to know:
  - Type A objects can be connected to type B objects
  - Don't need to know exact specifics of A and B
- Implications
  - Each component is required to understand a particular protocol
  - Interfaces between components defined by protocol – only need to understand external interfaces of components
  - Less flexible
  - Information passed to application must be explicitly passed

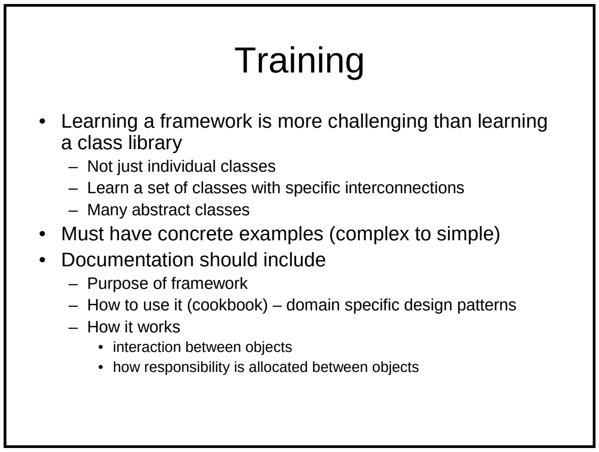
Graybox

- Define new concrete subclasses and use them to build application
- Subclasses are tightly coupled to super classes
- Requires more explicit knowledge about abstract classes
- Subclasses must meet specifications implied by super class
- Programmers must understand framework's interface in detail



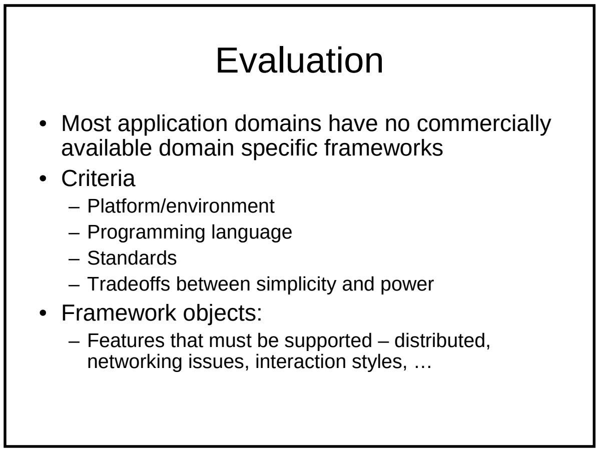
Whitebox Frameworks

- Program skeleton
  - – Subclasses are the additions to the skeleton
- Change the abstract classes that form the core of the framework – add new operators and/or attributes
- Requires the actual source code of framework (versus just the interface)
- Implications
  - – Framework implementation must be understood to use it
  - – Every application requires the creation of many new subclasses
  - – Can be difficult to learn – need to know hierarchical structure
  - – State of each instance is implicitly available to all methods in framework
  - – Changes to abstract classes can break existing concrete classes



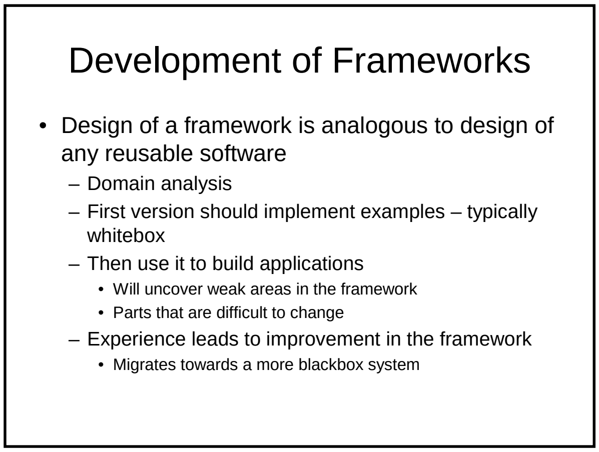
Training

- Learning a framework is more challenging than learning a class library
  - – Not just individual classes
  - – Learn a set of classes with specific interconnections
  - – Many abstract classes
- Must have concrete examples (complex to simple)
- Documentation should include
  - – Purpose of framework
  - – How to use it (cookbook) – domain specific design patterns
  - – How it works
    - interaction between objects
    - how responsibility is allocated between objects



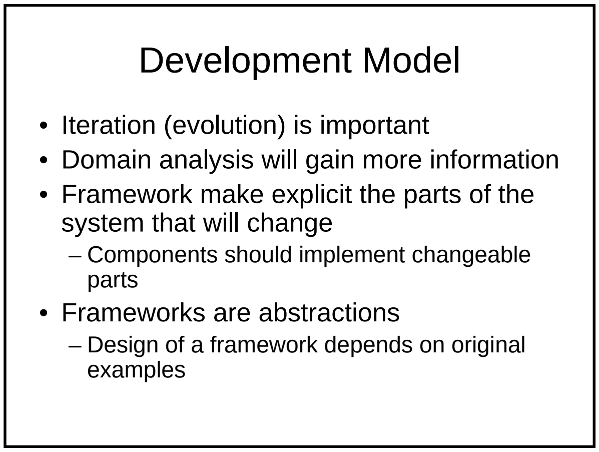
Evaluation

- Most application domains have no commercially available domain specific frameworks
- Criteria
  - – Platform/environment
  - – Programming language
  - – Standards
  - – Tradeoffs between simplicity and power
- Framework objects:
  - – Features that must be supported – distributed, networking issues, interaction styles, …



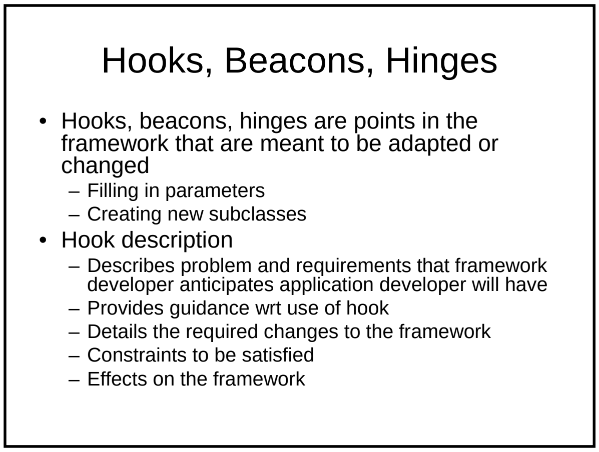
Development of Frameworks

- Design of a framework is analogous to design of any reusable software
  - Domain analysis
  - First version should implement examples – typically whitebox
  - Then use it to build applications
    - Will uncover weak areas in the framework
    - Parts that are difficult to change
  - Experience leads to improvement in the framework
    - Migrates towards a more blackbox system



Development Model

- Iteration (evolution) is important
- Domain analysis will gain more information
- Framework make explicit the parts of the system that will change
  - Components should implement changeable parts
- Frameworks are abstractions
  - Design of a framework depends on original examples



Hooks, Beacons, Hinges

- Hooks, beacons, hinges are points in the framework that are meant to be adapted or changed
  - Filling in parameters
  - Creating new subclasses
- Hook description
  - Describes problem and requirements that framework developer anticipates application developer will have
  - Provides guidance wrt use of hook
  - Details the required changes to the framework
  - Constraints to be satisfied
  - Effects on the framework

Hooks Adapt Framework

- Enabling/Disabling a feature
- Replacing a feature
- Augmenting a feature
- Adding a feature

Benefits of Frameworks

- Modularity
  - Encapsulate volatile implementation details behind stable interfaces
  - Localize impact of design and implementation changes
- Reusability
  - Stable interfaces enhance reusability of generic components
  - Leverages domain knowledge and prior experience

Benefits

- Extensibility
  - Hook methods allow applications to extend its stable interfaces
  - Hook methods decouple stable interfaces and behaviors of an application domain
- Inversion of Control
  - Application processing customized by event handler objects invoked via framework’s reactive dispatching mechanism
  - Allow framework rather than each application to determine which set of application specific methods to invoke in response to external events
    - Window messages from end users
    - Packets arriving on communications ports

Trade offs

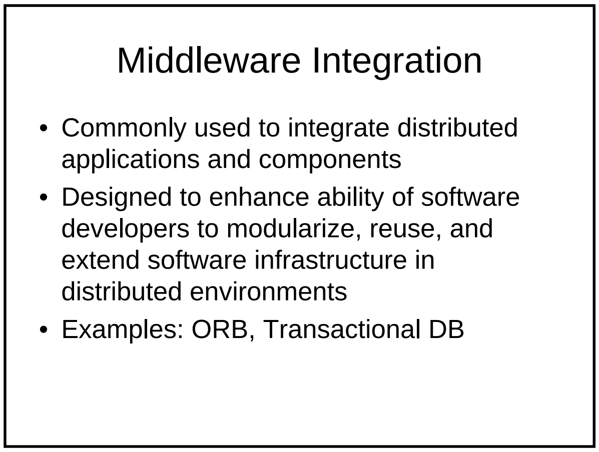
- Benefits of frameworks
  - Enable direct reuse of code
  - Enable large amounts of reuse vs standalone functions/classes
- Drawbacks
  - High initial learning curve
  - Flow of control for reactive dispatching is often non-intuitive
  - Verification/validation of generic components is often quite difficult

Classification of Frameworks

- System infrastructure
- Middleware integration
- Enterprise application

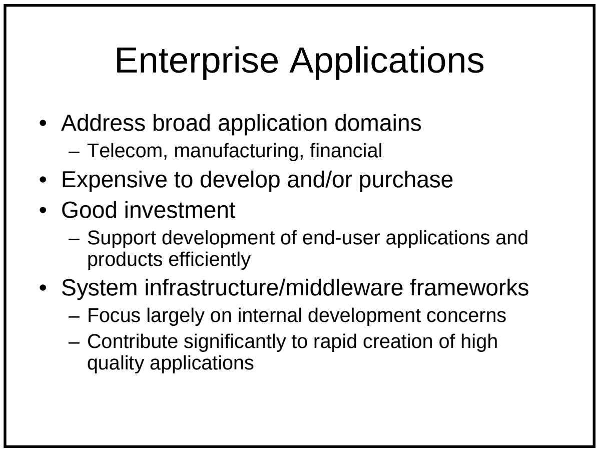
System Infrastructure

- Simplify development of portable and efficient system infrastructure
- Examples: UI and language processing tools
- Primarily used internally within a software development organization



Middleware Integration

- Commonly used to integrate distributed applications and components
- Designed to enhance ability of software developers to modularize, reuse, and extend software infrastructure in distributed environments
- Examples: ORB, Transactional DB



Enterprise Applications

- Address broad application domains
  - Telecom, manufacturing, financial
- Expensive to develop and/or purchase
- Good investment
  - Support development of end-user applications and products efficiently
- System infrastructure/middleware frameworks
  - Focus largely on internal development concerns
  - Contribute significantly to rapid creation of high quality applications