Object-Oriented Design

THE PROCESS AND THE PRAGMATICS

- Round-Trip Gestalt Design
- Process of OOD
- The Spiral Model
- Steps of OOD
- Pragmatics

7 - 1

Objectives of Module 7

- Present and discuss the process of Object-Oriented Design, identifying the activities and products produced during the four steps:
  - Identifying Classes and Objects
  - Identifying the Semantics of Classes and Objects
  - Identifying the Relationships Among Classes and Objects
  - Implementing Classes and Objects
- Present and discuss the pragmatics of Object-Oriented Design, including the experiences to be encountered on a live OOD project and the resources required
- Review the benefits and risks of Object-Oriented Design and present and discuss ways to transition from other design methodologies to an Object-Oriented Design methodology in a pragmatic fashion

ROUND-TRIP GESTALT DESIGN

Round-Trip Gestalt Design **- the style of design which emphasizes the incremental and iterative development of a system through the refinement of different yet consistent logical and physical views of the system as a whole**

Booch suggests that Round-Trip Gestalt Design is the foundation of the process of Object-Oriented Design.

Object-Oriented Design:

- **is an unconstrained and fuzzy process**
- **is a creative process, and creativity cannot be dictated by the definition of a few steps to follow or a few products to create**
- **depends upon a comprehensive understanding of the problem to be solved, as does any design process**

- **Round-Trip Gestalt Design**
- *Process of OOD*
- *The Spiral Model*
- *Steps of OOD*
- *Pragmatics*

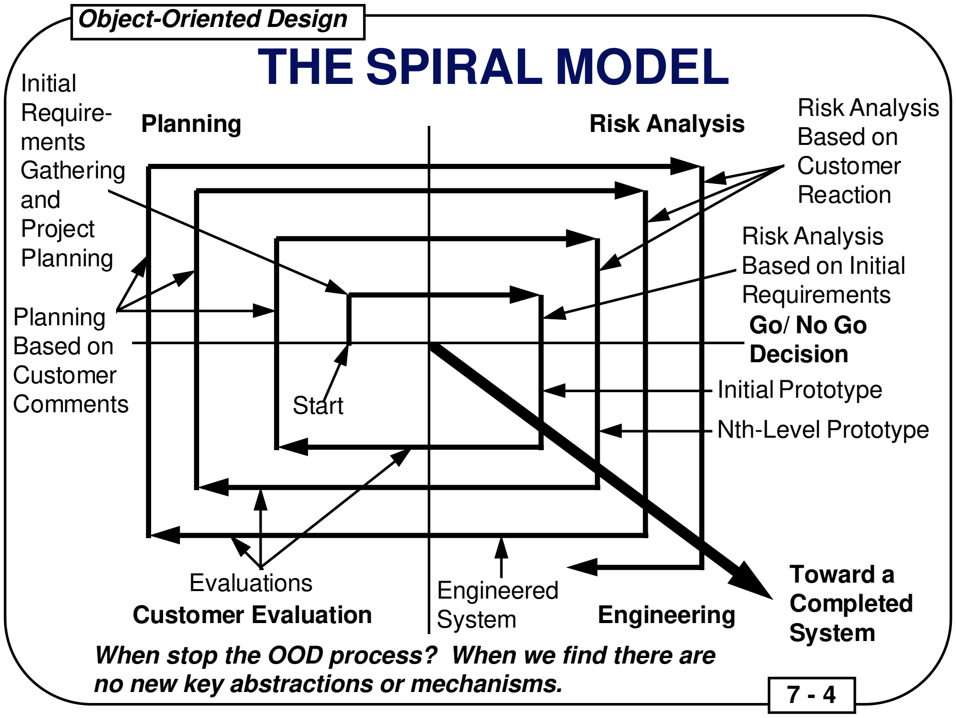
PROCESS OF OOD

OOD is an incremental, iterative process which generally tracks the following order of events during each iteration:

- Identify the classes and objects at a given level of abstraction.
- Identify the semantics of these classes and objects.
- Identify the relationships among these classes and objects.
- Implement these classes and objects.

These steps are part of an evolutionary development consistent with Barry Boehm's sprial model of software development.

- *Round-Trip Gestalt Design*
- **Process of OOD**
- **The Spiral Model**
- *Steps of OOD*
- *Pragmatics*



Iterative Refinement

First Loop

- Start at the center of the spiral; plan the project and gather initial requirements
- Perform a risk analysis based on these initial requirements; make a go/no go decision; continue if go
- Create an initial prototype of the system
- Customer (and developer) evaluate the prototype

Second Loop

- Feedback from the evaluation is used to refine the requirements and more project planning is done
- Perform a second risk analysis based on the revised requirements; make a go/no go decision; continue if go
- Create a second prototype, based either on the initial prototype or built from scratch
- Customer (and developer) evaluate the second prototype

Nth Loop

- Repeat the Second Loop as desired

After Last Go/No Go Decision

- Engineer the system

STEPS OF OOD

As mentioned earlier, these steps are:

- **Identify the classes and objects at a given level of abstraction.**
- **Identify the semantics of these classes and objects.**
- **Identify the relationships among these classes and objects.**
- **Implement these classes and objects.**

The rest of this module examines these steps, their key activities, and the possible products produced by them.

1. Identifying Classes and Objects

Two activities:

- **Discover the key abstractions of the problem space (the significant classes and objects).**
- **Invent the important mechanisms that provide the behavior required of objects that work toegether to achieve some function.**

Module 5 presented techniques which may be employed to find these abstractions and mechanisms.

Possible Products:

- **A listing of the names of significant classes and objects, using meaningful names that imply their semantics**
- **A specification of these abstractions and mechanisms by filling out the appropriate class and object templates**
- **Diagrams showing these abstractions and mechanisms**

- *Round-Trip Gestalt Design*
- *Process of OOD*
- *The Spiral Model*
- **Steps of OOD**
  - **1. Identifying Classes and Objects**
  - *2. Identifying the Semantics of Classes and Objects*
  - *3. Identifying the Relationships Among Classes and Objects*
  - *4. Implementing Classes and Objects*
- *Pragmatics*

2. Identifying the Semantics of Classes and Objects

One basic activity:

- Establish the meanings of the classes and objects identified in the previous step.

Possible Products:

- Refined templates based on those drafted in the previous step
- Documents of all the static and dynamic semantics of each key abstraction and mechanism
- New object diagrams that document any mechanisms invented
- Prototyped parts of the design that are used to analyze the current design and evaluate alternate approaches to subproblems considered to be areas of high risk

3. Identifying the Relationships Among Classes and Objects

Key activities:

- **Discover patterns among classes, used to reorganize and simplify the class structure, and patterns among cooperative collections of objects, used to generalize the mechanisms already embodied in the design.**
- **Make visibility decisions, such as how classes see one another, how objects see one another, and what classes and objects should not see one another.**
- **Refine the protocols of various classes from earlier steps.**

Possible Products:

- **Completed versions of most of the logical models of the design**
- **Refined and complete class and object diagrams**
- **The essential module diagrams of the system**
- **New prototypes and/or refined older prototypes**

- *Round-Trip Gestalt Design*
- *Process of OOD*
- *The Spiral Model*
- **Steps of OOD**
  - *1. Identifying Classes and Objects*
  - *2. Identifying the Semantics of Classes and Objects*
  - **3. Identifying the Relationships Among Classes and Objects**
  - *4. Implementing Classes and Objects*
- *Pragmatics*

4. Implementing Classes and Objects

Key activities:

- Make design decisions concerning the representation of the classes and objects invented.
- Allocate classes and objects to modules and programs to processors.
- Decide how the behavior of each class and module should be implemented.

Possible Products:

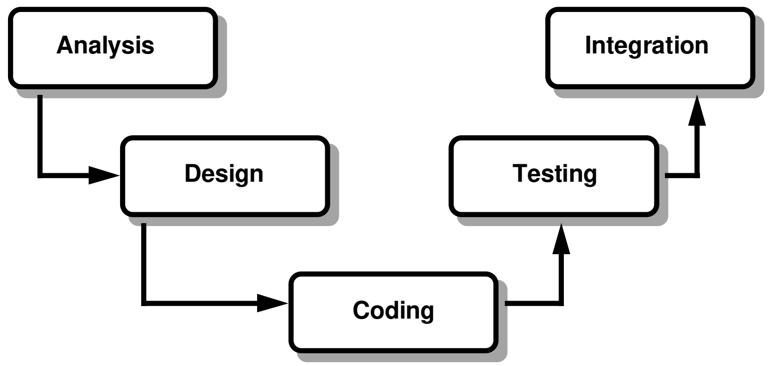
- The concrete interface of each class and object important at the current level of abstraction
- The refined class structure of the system
- The implementation of each important class template at the current level of abstraction
- Complete versions of all diagrams

We are now ready to start over with Step 1 of the next level of abstraction, continuing our iteration through the Spiral Model.

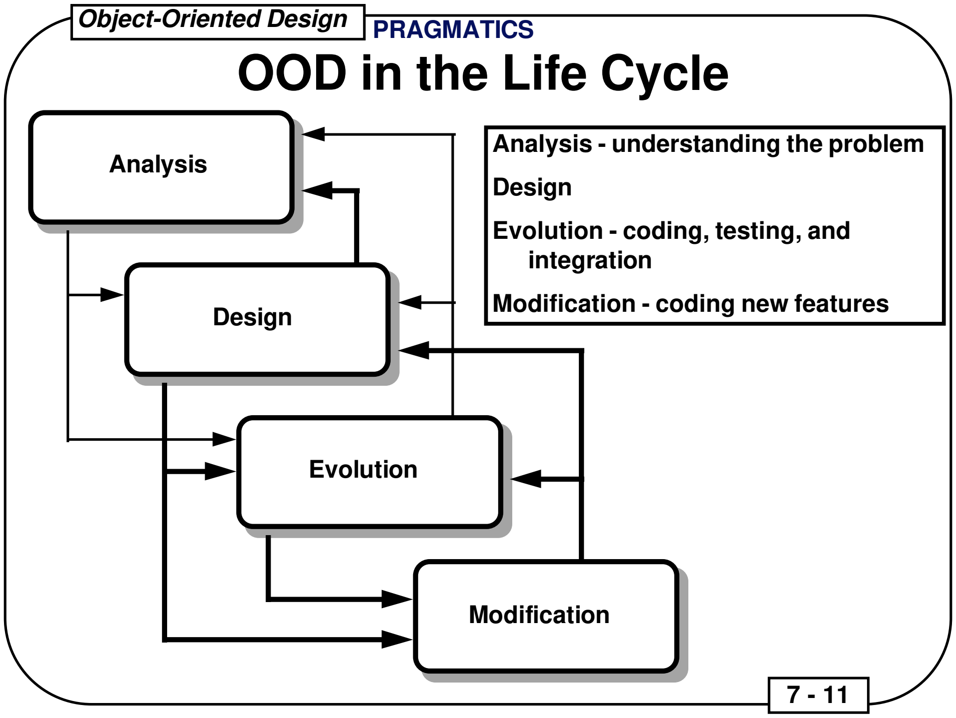
- *Round-Trip Gestalt Design*
- *Process of OOD*
- *The Spiral Model*
- **Steps of OOD**
  - *1. Identifying Classes and Objects*
  - *2. Identifying the Semantics of Classes and Objects*
  - *3. Identifying the Relationships Among Classes and Objects*
  - **4. Implementing Classes and Objects**
- *Pragmatics*

PRAGMATICS

Module 1 of this courses presented an overview of software engineering, covering topics such as the waterfall model, the software development life cycle, and other such items of general pragmatic interest. An understanding of these topics will be assumed in this presentation, and the emphasis will be on the integration of OOA and OOD into a software engineering practice.

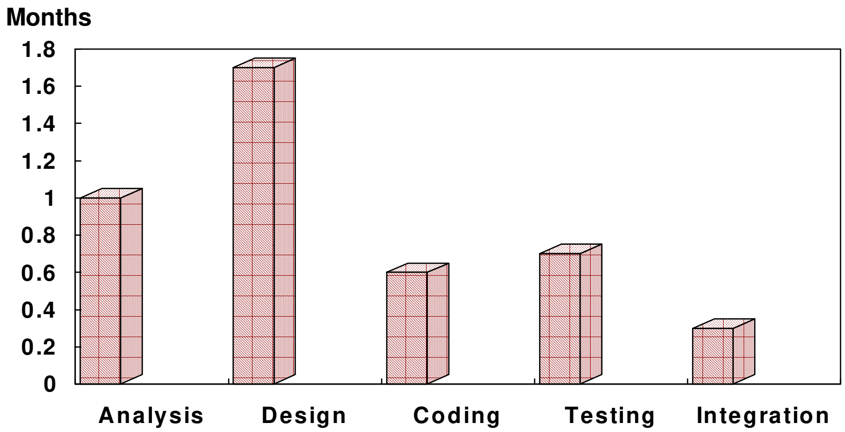


- *Round-Trip Gestalt Design*
- *Process of OOD*
- *The Spiral Model*
- *Steps of OOD*
- **Pragmatics**
  - *OOD in the Life Cycle*
  - *Resource Allocation with OOD*
  - *Development Team Skills*
  - *Milestones and Products*
  - *Quality Assurance*
  - *Tool Support*
  - *Benefits and Risks of OOD*
  - *Transition to OOD*



- *Round-Trip Gestalt Design*
- *Process of OOD*
- *The Spiral Model*
- *Steps of OOD*
- **Pragmatics**
  - **OOD in the Life Cycle**
  - *Resource Allocation with OOD*
  - *Development Team Skills*
  - *Milestones and Products*
  - *Quality Assurance*
  - *Tool Support*
  - *Benefits and Risks of OOD*
  - *Transition to OOD*

Resource Allocation with OOD



Design is clearly the most time-consuming activity during an Object-Oriented development

- *Round-Trip Gestalt Design*
- *Process of OOD*
- *The Spiral Model*
- *Steps of OOD*
- **Pragmatics**
  - *OOD in the Life Cycle*
  - **Resource Allocation with OOD**
  - *Development Team Skills*
  - *Milestones and Products*
  - *Quality Assurance*
  - *Tool Support*
  - *Benefits and Risks of OOD*
  - *Transition to OOD*

Development Team Skills

With OOD, four kinds of developers are needed:

- System architects
- Class designers
- Class implementers
- Application programmers

Additionally, two specific product-independent positions are needed:

- a class librarian, who maintains configuration and version control of the classes for one or more projects so that such a library is useful and does not become a vast wasteland of junk classes and a proactive step can be taken to encourage software reuse
- a toolsmith, who creates domain-specific tools and tailors existing tools for a project (including a common test scaffolding) for current and future projects

The best results are obtained with fewer and better people.

Milestones

Managers need to be kept informed of the progress of the effort, and, with an incremental and iterative approach, it is difficult to lay out traditional milestones. The best solution is to carry out a few formal and many informal design reviews:

- In the earliest design reviews, developers, domain experts, and users should review the draft of the class structure of the design and its basic mechanisms.
- As these key abstractions and mechanisms mature, later design reviews can cover the products of the logical and physical design of the system in more detail.
- The worst way to measure progress is by measuring the lines of code produced.
- A better way to measure progress is by counting the classes in the logical design and the modules in the physical design that are working.
- Another better way to measure progress is to measure the stability of the key abstractions (how often they change).

Products

The products of object-oriented design include:

- sets of class diagrams
- sets of object diagrams
- sets of module diagrams, which denote implementations of some combination of classes and objects
- sets of process diagrams, which denote programs
- code
- design document, where each major software segments should have its own design document that includes:
  - required functions
  - class and object diagrams and templates
  - module and process diagrams and templates
  - results of executable prototypes

Quality Assurance

Since we are concerned with created software which is of quality, it is often wise to place into effect certain activities to help meet this objective:

- Peer reviews (walkthroughs) of designs with people not involved in the project
- Unit testing, performed by someone other than the developer who creates a test scaffolding for each class or object
- Integration testing, which is performed by the team using a test scaffolding to test whole mechanisms at a time and the system as a whole

Tool Support

In OOD, tools to support the activity must focus on more than the development of just code. OOD highlights key abstractions and mechanisms, so our tools need to be able to capture their semantics as well. Six kinds of tools have been identified for use with OOD:

- a graphics-based system supporting the object-oriented design notation used by the organization and allowing the users to reverse engineer the code into the diagrams after the code has been changed during the evolution of the system
- a browser which can be used to easily move through the class and module architectures of the system
- an incremental compiler
- a debugger that knows about class and object semantics
- configuration management and version control tools
- a class librarian which can provide ready access to a growing library of domain-specific reusable software components

Benefits of OOD

Early adopters of new technologies do so for two main reasons:

- They seek a competative advantage (reduced time to market, greater product flexibility, schedule predictability, etc.) through new techniques.
- They have complex problems that don't seem to have any other solution.

OOD is not a new technology, but it is not exactly a mature technology either. For now, many developers take it on faith that there are considerable benefits to be gained from the use of OOD. Some professed benefits include:

- OOD exploits the expressive power of all object-based and object-oriented programming languages
- OOD encourages the reuse of software components
- OOD leads to systems more resilient to change
- OOD reduces development risk
- OOD appeals to the working of human cognition

Risks of OOD

There are to main areas of risk with OOD: performance and start-up costs.

Performance overhead comes from sources such as:

- **message passing between objects, with dynamic lookups taking perhaps 1.75 to 2.5 times as long as simple subprogram calls; on the plus side, studies indicate that dynamic lookup is needed in only about 20% of most method invocations and static lookup (normal subprogram calls) can be used for the rest**
- **a glut of method invocations, much more than the normal set of subprogram calls found in conventional designs, due to the layering of the object-oriented software; such layering is needed for the understandability of the system, but inline code may be used or the layering may be removed for implementation**
- **a large number of classes based on various superclasses may result in an extensive duplication of code from the superclasses; advancements in compiler technology will eventually reduce or eliminate this problem**

Risks of OOD, Continued

Performance risks, continued:

- the paging behavior of running programs may require code to be segmented, where code in a segment takes less space if subroutine calls are made to other code within a segment than to other code in a different segment, and large classes or classes declared in separate files may find their code scattered over several segments
- dynamic allocation and destruction of objects usually requires more run time overhead to use the heap than the allocation and destruction of objects on the stack, and most object-oriented constructs for dynamic allocation use the heap; this can be solved by completing the dynamic creation of objects as part of the elaboration of the program rather than during any time-critical operation, but a change in compiler technology is required

Risks of OOD, Continued

Start-up costs associated with object-oriented design may be a major barrier to adopting OOD:

- the capitalization of software development tools may be extreme
- the first time through, there will be no domain-specific software to employ as a basis of reuse for many application domains
- using OOD for the first time will usually fail without the appropriate training
- it takes time to develop the proper mindset for OOD, and this new way of thinking must be embraced by both developers and managers

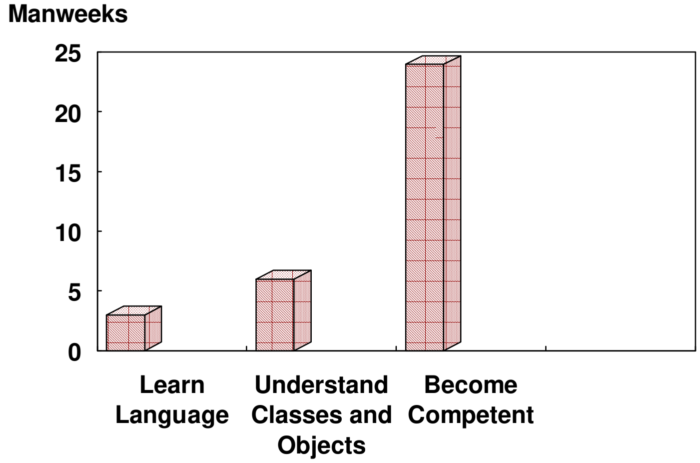
Transition to OOD

The change of mindset to OOD is perhaps the main stumbling block. The following is recommended:

- Provide formal training to both developers and managers in the elements of the object model.
- Use OOD in a low-risk project first, and allow the team to make mistakes. Good candidate projects include software development tools or domain-specific class libraries which can be used as resources in later projects.
- Use team members from this low-risk project to seed other projects and act as mentors for the object-oriented approach.
- Expose the developers and managers to examples of well-structured object-oriented systems.

Transition to OOD - Timeline

Booch quotes something like the following from his experiences:



Learning by example is perhaps the most efficient and effective approach.