Design Engineering

Module 4

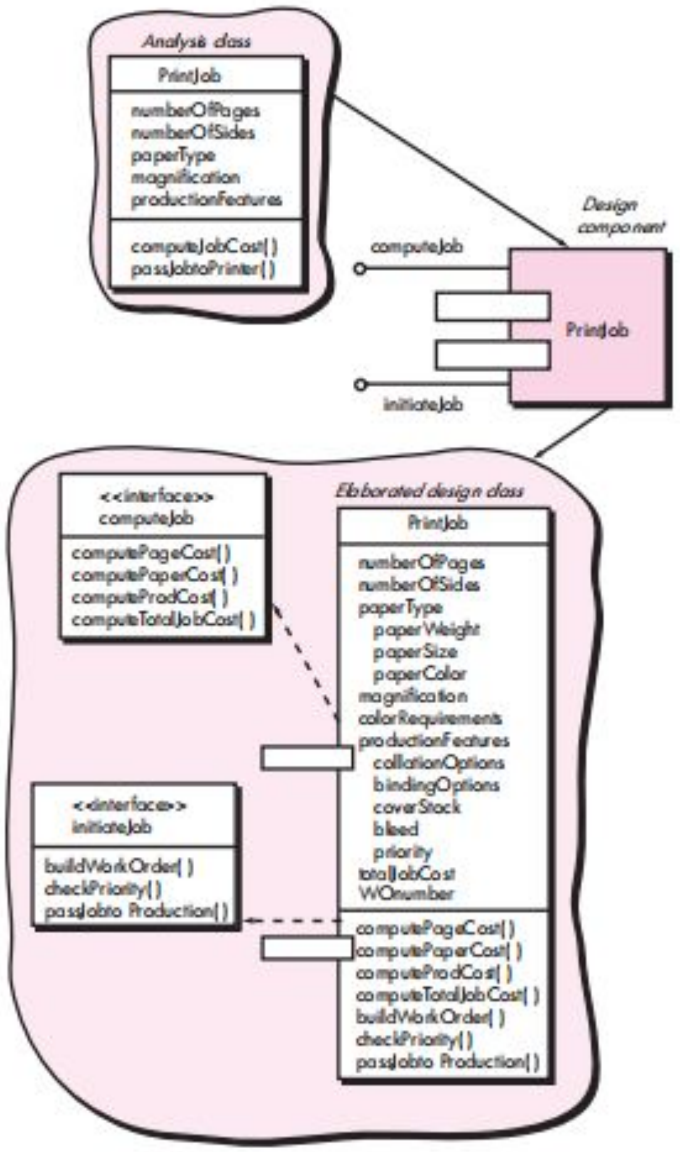
Design Engineering

Component Level Design
Component-level design occurs after the first iteration of architectural design has been completed. At this stage, the overall data and program structure of the software has been established. The intent is to translate the design model into operational software.

Component
A component is a modular building block for computer software. More formally, the OMG Unified Modeling Language Specification [OMG03a] defines a component as "... a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces." components populate the software architecture
and, as a consequence, play a role in achieving the objectives and requirements of the system to be built. Because components reside within the software architecture, they must communicate and collaborate with other components and with entities (e.g., other systems, devices, people) that exist outside the boundaries of the software.

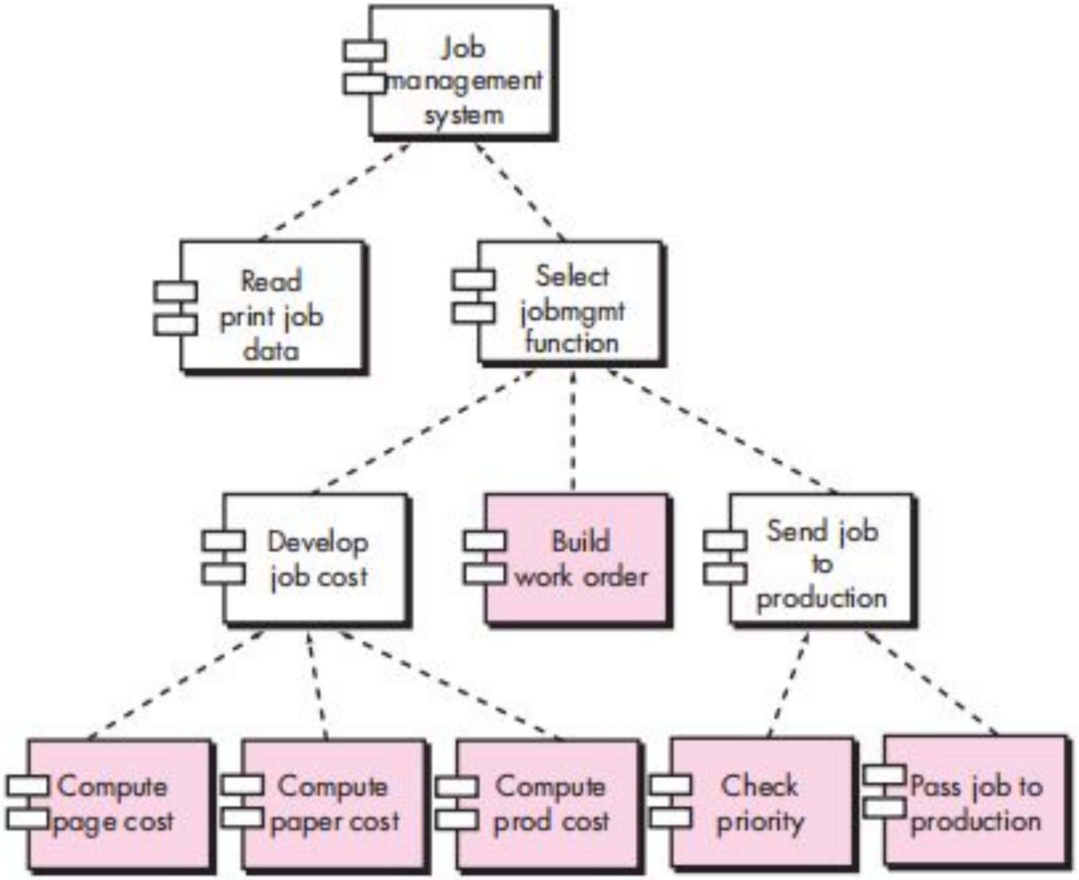
Design Engineering

Objet Oriented Design Component



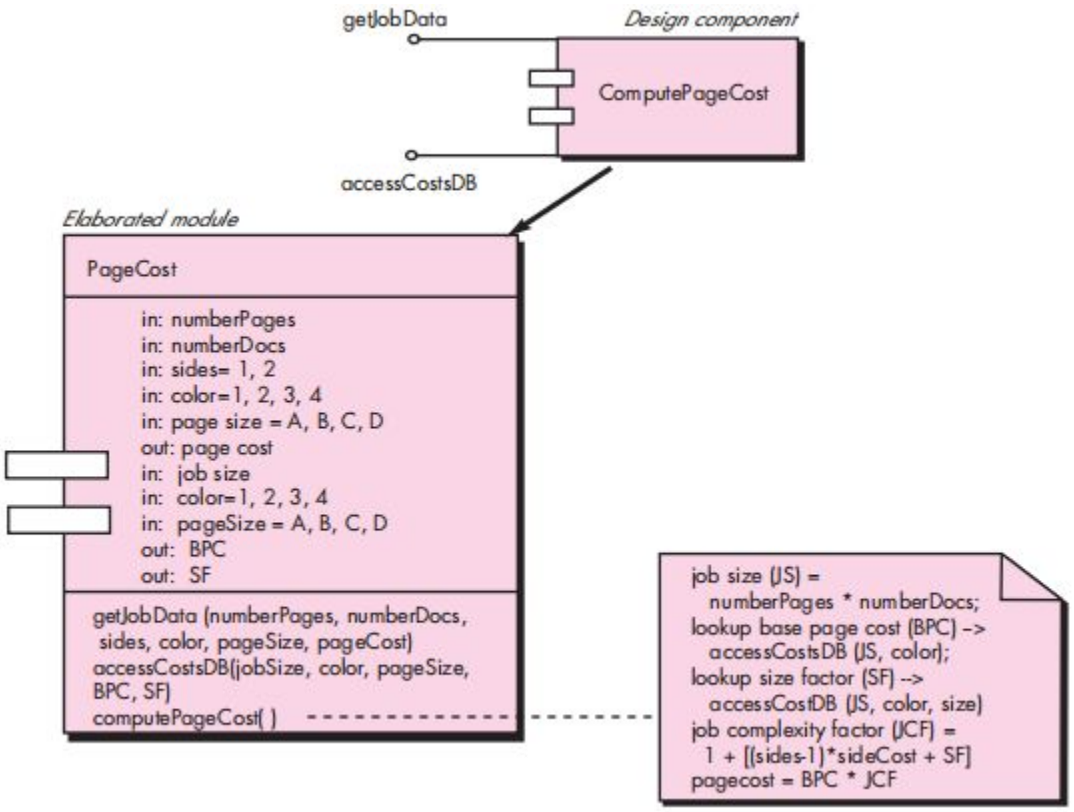
Design Engineering

Component Design for traditional System



Design Engineering

Component Design using modified UML


getJobData
Design component
ComputePageCost
accessCostsDB
Elaborated module
PageCost
in: numberPages
in: numberDocs
in: sides= 1, 2
in: color=1, 2, 3, 4
in: page size = A, B, C, D
out: page cost
in: job size
in: color=1, 2, 3, 4
in: pageSize = A, B, C, D
out: BPC
out: SF
getJobData (numberPages, numberDocs, sides, color, pageSize, pageCost)
accessCostsDB(jobSize, color, pageSize, BPC, SF)
computePageCost( )
job size (JS) = numberPages * numberDocs;
lookup base page cost (BPC) --> accessCostsDB (JS, color);
lookup size factor (SF) --> accessCostDB (JS, color, size)
job complexity factor (JCF) = 1 + [(sides-1)*sideCost + SF]
pagecost = BPC * JCF

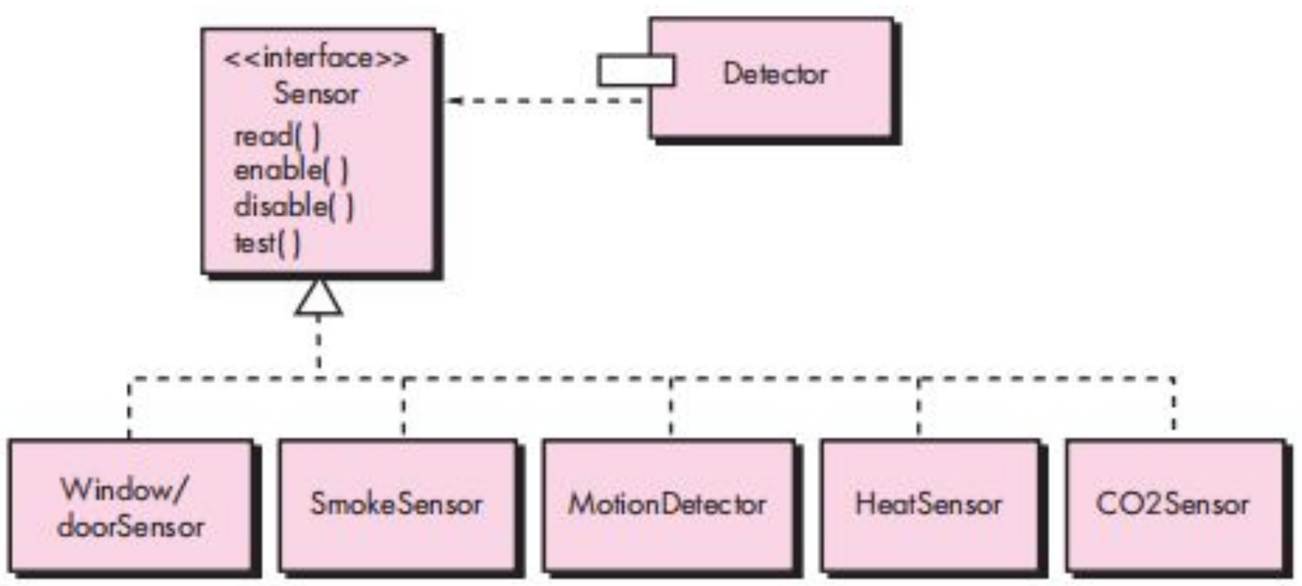
Design Engineering

DESIGNING CLASS -BASED COMPONENTS

Four basic design principles are applicable to component-level design and have been widely adopted when object-oriented software engineering is applied. The underlying motivation for the application of these principles is to create designs that are more amenable to change and to reduce the propagation of side effects when changes do occur.

Design Engineering

The Open-Closed Principle (OCP).



Design Engineering

- The Liskov Substitution Principle (LSP).
- Dependency Inversion Principle (DIP).
- The Interface Segregation Principle (ISP).

Although component-level design principles provide useful guidance, components themselves do not exist in a vacuum. In many cases, individual components or classes are organized into subsystems or packages. It is reasonable to ask how this packaging activity should occur. Exactly how should components be organized as the design proceeds? Martin [Mar00] suggests additional packaging principles that are applicable to component-level design:

- The Release Reuse Equivalency Principle (REP).
- The Common Closure Principle (CCP).
- The Common Reuse Principle (CRP).

Design Engineering

Component-Level Design Guidelines:
In addition to the principles discussed in Section 10.2.1, a set of pragmatic design guidelines can be applied as component-level design proceeds. These guidelines apply to components, their interfaces, and the dependencies and inheritance characteristics that have an impact on the resultant design. Ambler [Amb02b] suggests the following guidelines:

- Components. Naming conventions should be established for components that are specified as part of the architectural model and then refined and elaborated as partof the component-level model.
- Interfaces.
- Dependencies and Inheritance.

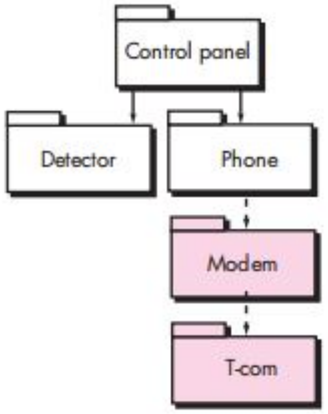
Design Engineering

Cohesion: It implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself. Lethbridge and Laganiére [Let01] define a number of different types of cohesion (listed in order of the level of the cohesion4):

- Functional. Exhibited primarily by operations, this level of cohesion occurs when a component performs a targeted computation and then returns a result.
- Communicational. All operations that access the same data are defined within one class. In general, such classes focus solely on the data in question, accessing and storing it

Design Engineering

Layer. Exhibited by packages, components, and classes, this type of cohesion occurs when a higher layer accesses the services of a lower layer, but lower layers do not access higher layers. Consider, for example, the SafeHome security function requirement to make an outgoing phone call if an alarm is sensed. It might be possible to define a set of layered packages as shown in Figure. The shaded packages contain infrastructure components. Access is from the control panel package downward.



Design Engineering

Classes and components that exhibit functional, layer, and communicational cohesion are relatively easy to implement, test, and maintain. You should strive to achieve these levels of cohesion whenever possible. It is important to note, however, that pragmatic design and implementation issues sometimes force you to opt for
lower levels of cohesion

Design Engineering

Coupling: Coupling is a qualitative measure of the degree to which classes are connected to one another. As classes (and components) become more interdependent, coupling increases. An important objective in component-level design is to keep coupling as low as is possible.

Class coupling can manifest itself in a variety of ways. Lethbridge and Laganiére [Let01] define the following coupling categories:

- Content coupling. Occurs when one component "surreptitiously modifies data that is internal to another component" [Let01]. This violates information hiding—a basic design concept.

Design Engineering

- Common coupling. Occurs when a number of components all make use of a global variable. Although this is sometimes necessary (e.g., for establishing default values that are applicable throughout an application), common coupling can lead to uncontrolled error propagation and unforeseen side effects when changes are made.

- Control coupling. Occurs when operation A() invokes operation B() and passes a control flag to B. The control flag then "directs" logical flow within B. The problem with this form of coupling is that an unrelated change in B can result in the necessity to change the meaning of the control flag that A passes. If this is overlooked, an error will result.

Design Engineering

- Stamp coupling. Occurs when ClassB is declared as a type for an argument of an operation of ClassA. Because ClassB is now a part of the definition of ClassA, modifying the system becomes more complex.
- Data coupling. Occurs when operations pass long strings of data arguments. The “bandwidth” of communication between classes and components grows and the complexity of the interface increases. Testing and maintenance are more difficult.
- Routine call coupling. Occurs when one operation invokes another. This level of coupling is common and is often quite necessary. However, it does increase the connectedness of a system.

Design Engineering

- Type use coupling. Occurs when component A uses a data type defined incomponent B (e.g., this occurs whenever “a class declares an instance variable or a local variable as having another class for its type” [Let01]). If the type definition changes, every component that uses the definition must also change.

- Inclusion or import coupling. Occurs when component A imports or includes a package or the content of component B.

- External coupling. Occurs when a component communicates or collaborates with infrastructure components (e.g., operating system functions,database capability, telecommunication functions). Although this type of coupling is necessary, it should be limited to a small number of components or classes within a system.

Design Engineering

CONDUCTING COMPONENT-LEVEL DESIGN

The following steps represent a typical task set for component-level design, when it is applied for an object-oriented system.

Step 1. Identify all design classes that correspond to the problem domain.

Step 2. Identify all design classes that correspond to the infrastructure domain.

Step 3. Elaborate all design classes that are not acquired as reusable components.

Step 3a. Specify message details when classes or components collaborate

Step 3b. Identify appropriate interfaces for each component.

Step 3c. Elaborate attributes and define data types and data structures required to implement them.

Design Engineering

CONDUCTING COMPONENT-LEVEL DESIGN
Step 3d. Describe processing flow within each operation in detail.
Step 4. Describe persistent data sources (databases and files) and identify the classes required to manage them.
Step 5. Develop and elaborate behavioral representations for a class or component.
Step 6. Elaborate deployment diagrams to provide additional implementation detail.
Step 7. Refactor every component-level design representation and always consider alternatives.

Design Engineering

USER INTERFACE DESIGN:
User interface design creates an effective communication medium between a human and a computer. Following a set of interface design principles, design identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype.

Three golden rules:
1. Place the user in control.
2. Reduce the user's memory load.
3. Make the interface consistent.

These golden rules actually form the basis for a set of user interface design principles that guide this important aspect of software design.

Design Engineering

Place the User in Control:

- Define interaction modes in a way that does not force a user into unnecessary or undesired actions.
- Provide for flexible interaction.
- Allow user interaction to be interruptible and undoable.
- Streamline interaction as skill levels advance and allow the interaction to be customized.
- Hide technical internals from the casual user.
- Design for direct interaction with objects that appear on the screen.

Design Engineering

Reduce the User's Memory Load:

- Reduce demand on short-term memory.
- Establish meaningful defaults.
- Define shortcuts that are intuitive.
- The visual layout of the interface should be based on a real-world
- metaphor.
- Disclose information in a progressive fashion.

Design Engineering

Make the Interface Consistent:

- Allow the user to put the current task into a meaningful context.
- Maintain consistency across a family of applications.
- If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

Design Engineering

Interface design Steps:

Although many different user interface design models (e.g., [Nor86], [Nie00]) have been proposed, all suggest some combination of the following steps:

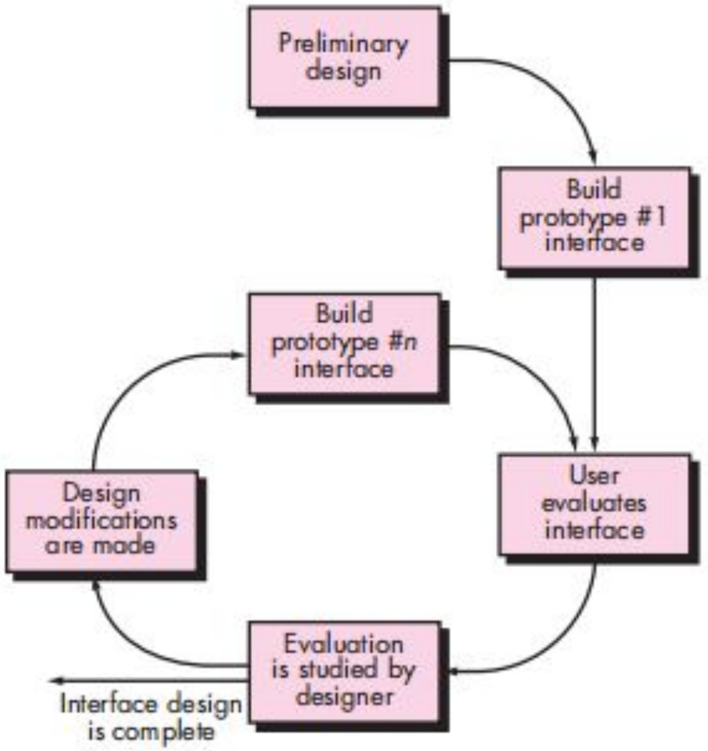
1. Using information developed during interface analysis, define interface objects and actions (operations).
2. Define events (user actions) that will cause the state of the user interface to change. Model this behavior.
3. Depict each interface state as it will actually look to the end user.
4. Indicate how the user interprets the state of the system from information provided through the interface.

Design Engineering

DESIGN EVALUATION:
Once you create an operational user interface prototype, it must be evaluated to determine whether it meets the needs of the user. Evaluation can span a formality spectrum that ranges from an informal “test drive,” in which a user provides impromptu feedback to a formally designed study that uses statistical methods for the evaluation of questionnaires completed by a population of end users.

Design Engineering

DESIGN EVALUATION:


Preliminary design
Build prototype #1 interface
Build prototype #n interface
User evaluates interface
Design modifications are made
Evaluation is studied by designer
Interface design is complete

Design Engineering

DESIGN EVALUATION:
The user interface evaluation cycle takes the form shown in Figure. After the design model has been completed, a first-level prototype is created.

The prototype is evaluated by the user,who provides you with direct comments about the efficacy of the interface. In addition, if formal evaluation techniques are used (e.g., questionnaires, rating sheets), you can extract information from these data (e.g., 80 percent of all users did not like the mechanism for saving data files).

Design modifications are made based on user input, and the next level prototype is created.

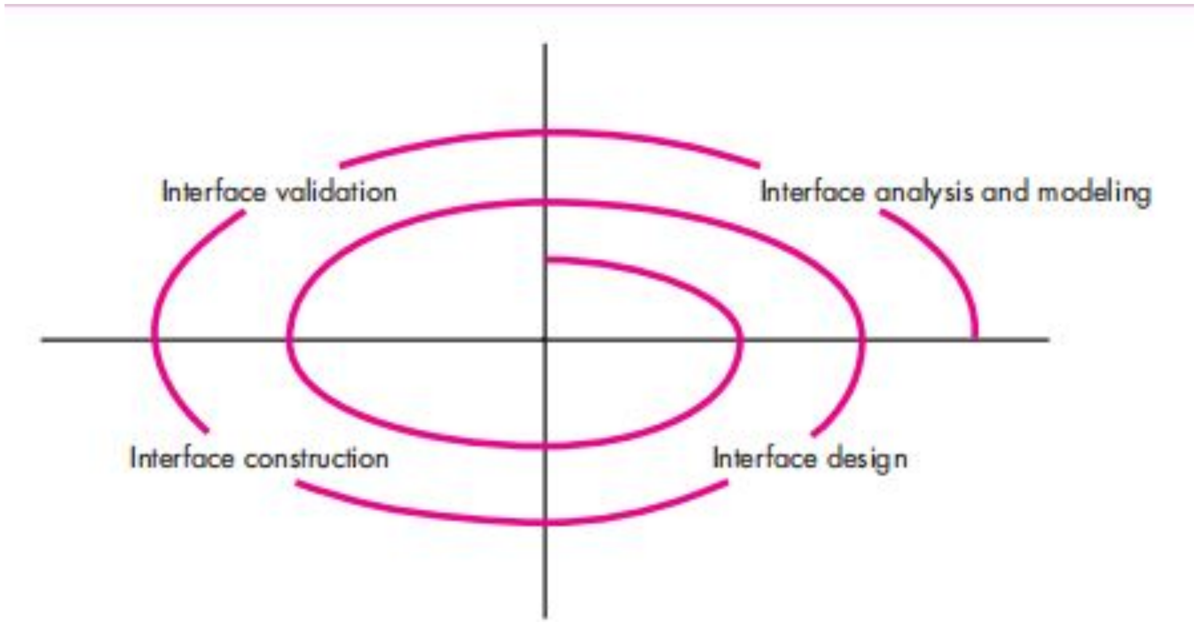
The evaluation cycle continues until no further modifications to the interface design are necessary.

Design Engineering

USER INTERFACE ANALYSIS AND DESIGN:
The overall process for analyzing and designing a user interface begins with the creation of different models of system function (as perceived from the outside). You begin by delineating the human- and computer-oriented tasks that are required to achieve system function and then considering the design issues that apply to all interface designs. Tools are used to prototype and ultimately implement the design model, and the result is evaluated by end users for quality.

Design Engineering

The Process



Design Engineering

INTERFACE ANALYSIS:

A key tenet of all software engineering process models is this: understand the problem before you attempt to design a solution. In the case of user interface design, understanding the problem means understanding
(1) the people (end users) who will interact with the system through the interface,
(2) the tasks that end users must perform to do their work,
(3) the content that is presented as part of the interface,
and
(4) the environment in which these tasks will be conducted. .

Design Engineering

INTERFACE ANALYSIS:

User Analysis:

- User Interviews.
- Sales input.
- Marketing input.
- Support input.

Task Analysis and Modeling:

- Use cases.
- Task elaboration.
- Object elaboration.
- Workflow analysis.
- Hierarchical representation.

Analysis of Display Content

Analysis of the Work Environment