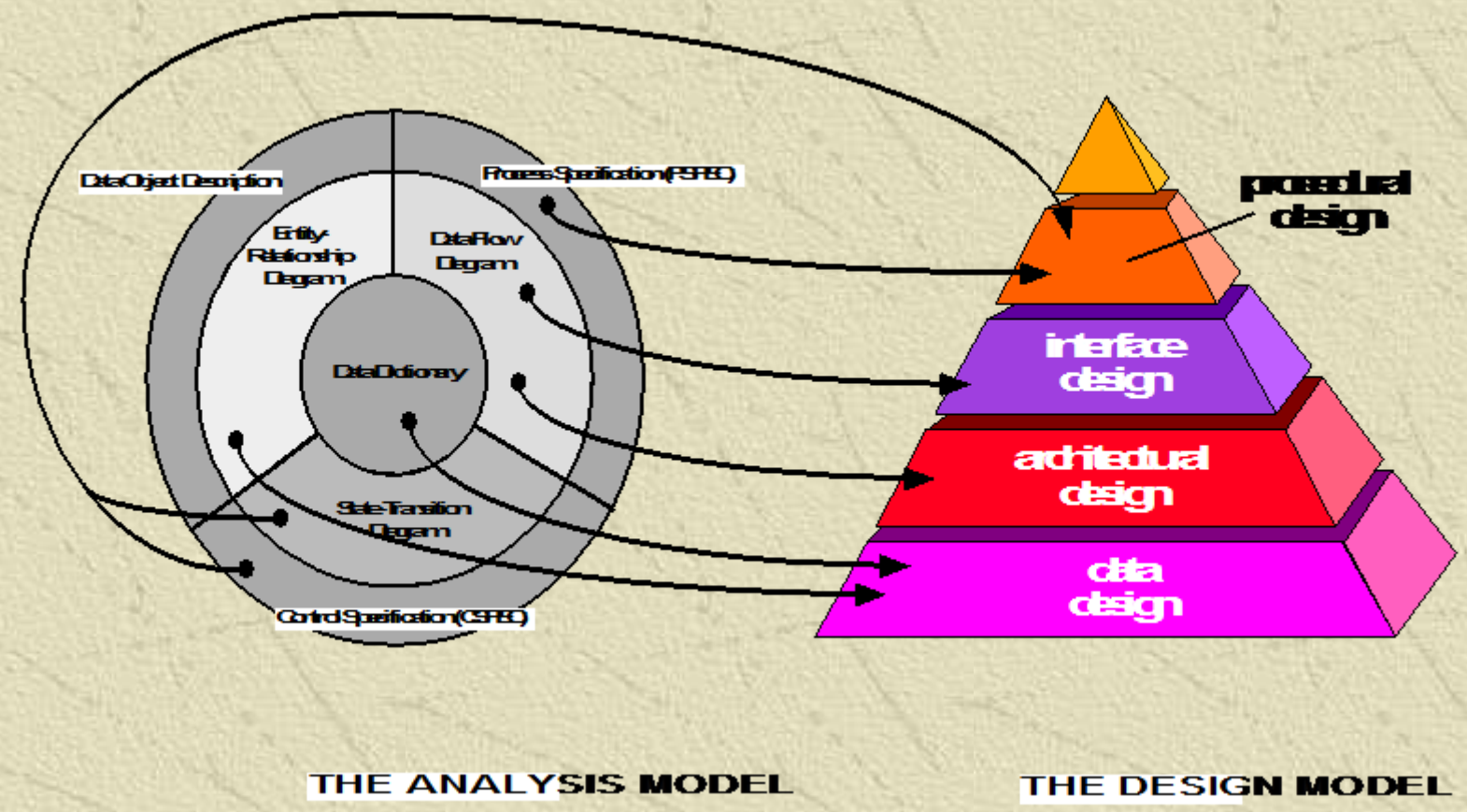
Design Concepts and Principles

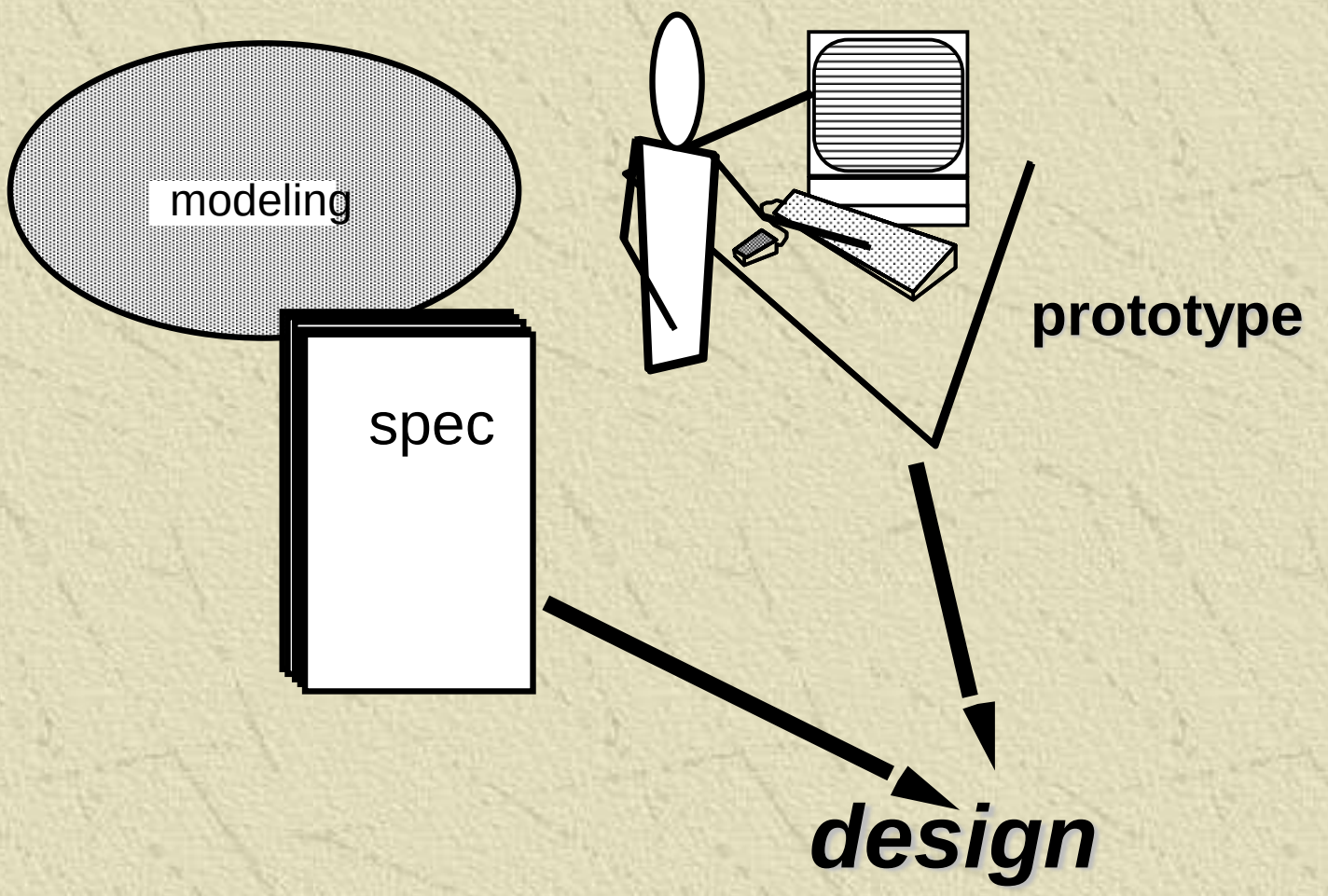
Why design?

- The design of the system is one of the most crucial stages:
  - "Low" enough to address implementation issues.
  - "High" enough to avoid wasting time.
- Problems solved at design time will save coding and debugging time; problems left open will become bugs.

Analysis to Design



Design: Where Do We Begin?



Design Guidelines

- **A design should exhibit a hierarchical organization that makes intelligent use of control among elements of software.**
- **A design should be modular – the software should be logically partitioned into elements that perform specific functions and sub-functions.**
- **A design should contain both data and procedural abstractions.**

Design Guidelines

- **A design should lead to modules that exhibit independent functional characteristics.**
- **A design should lead to interfaces that reduce the complexity of connections between modules and with the external environment.**

Design Principles

- **The design process should not suffer from "tunnel vision".**
- **The design should be traceable to the analysis model.**
- **The design should not reinvent the wheel.**
- **The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world.**
- **The design should exhibit uniformity and integration.**

Design Principles

- **The design should be structured to accommodate change.**
- **The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.**
- **Design is not coding, coding is not design.**
- **The design should be assessed for quality as it is being created, not after the fact.**
- **The design should be reviewed to minimize conceptual (semantic) errors.**

Key design concepts

- What are we trying to achieve?
  - Decomposition
  - Abstraction
  - Information hiding
  - (Correct) Modularity
  - Extensibility
  - “Virtual machines”
  - Hierarchy

Decomposition

- Large pieces of code are impossible to understand or maintain.
  - "Monster" classes
  - "Monster" methods
- How to avoid?
  - Decompose the system to modules until each module has a single major role.
  - Distribute responsibility among modules equally
    - No "show runners".

Decomposition rules

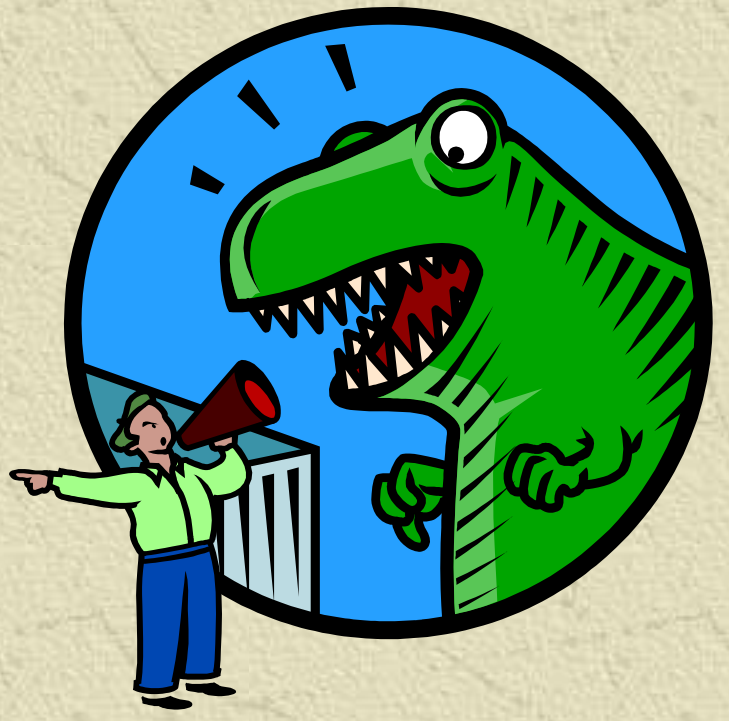
- The palm rule: a method containing logics should not be larger then your palm.
  - Usually a single editor screen
- The scroll rule: the entire class should be small enough to scroll through effortlessly.
  - Usually less then 1000 code lines, best less then 200.

Decomposition - example

- Consider the Simple Mail Transfer Protocol (SMTP) for sending and receiving e-mail.
  - Has a sending and receiving side.
  - The protocol has several stages (handshake, recipients list, the content itself…)
  - All communication is low-level (plain text) and has to be parsed
  - Each stage must be handled and acknowledged.
  - And so on…

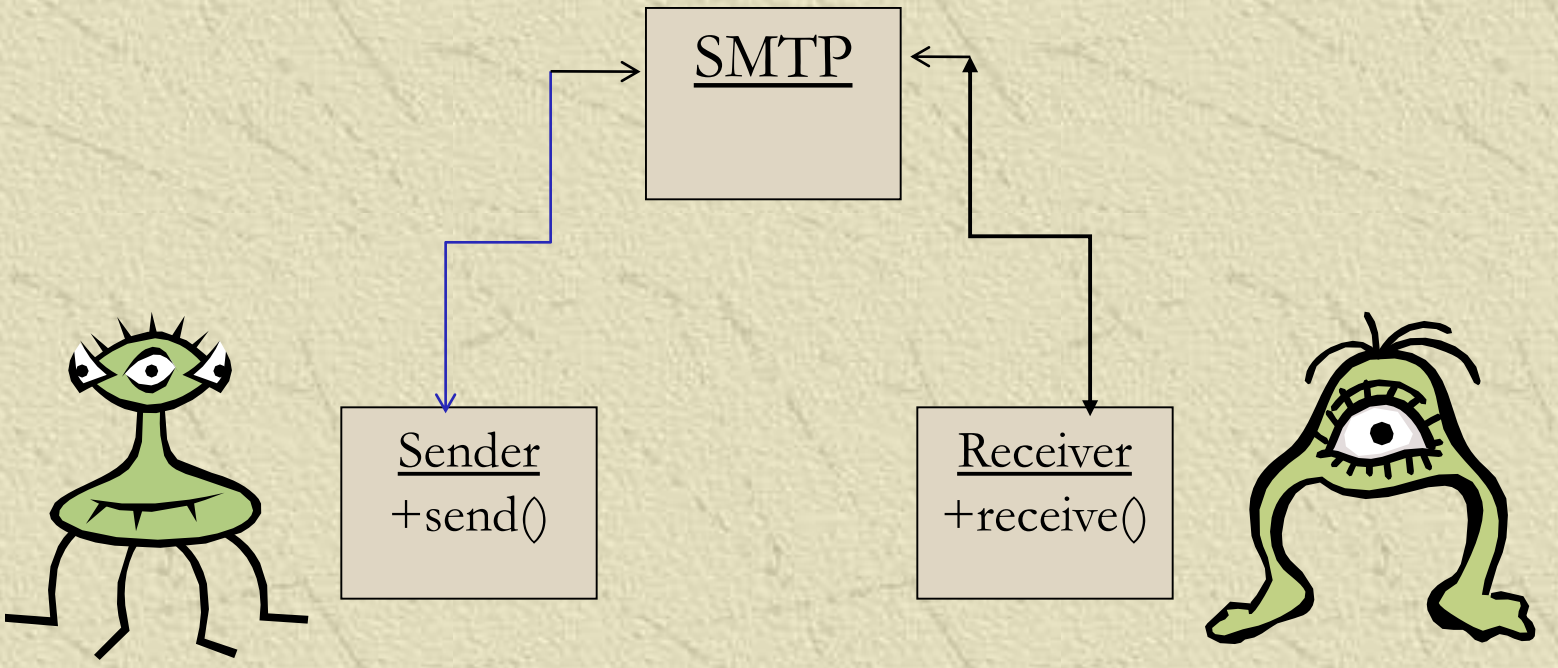
SMTP – phase 1

SMTP
+send()
+receive()



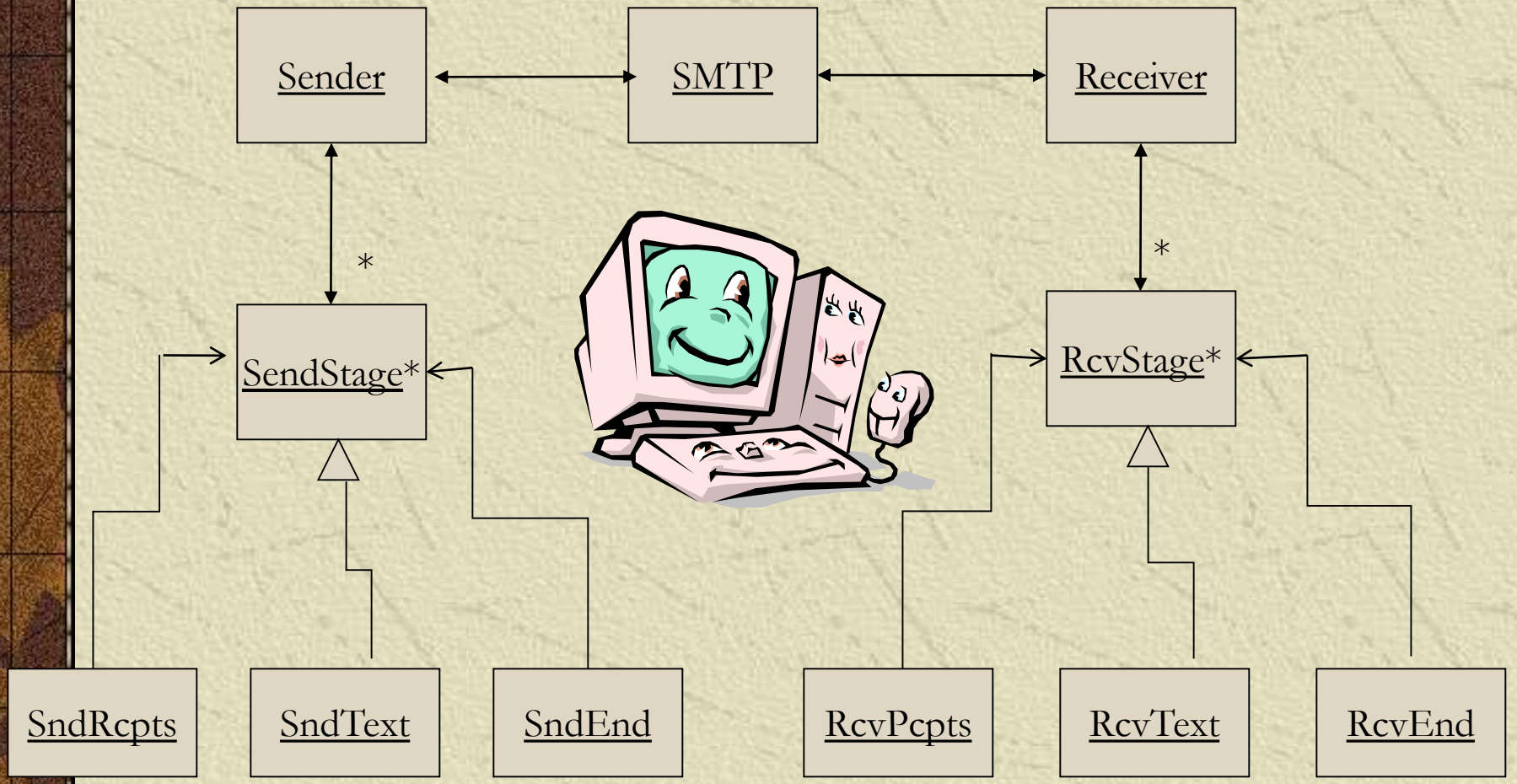
HELP!!!
Isn't that the class that ate Cincinnati…

SMTP – phase 2



Monsters, but small ones…

SMTP – phase 3


Sender
SMTP
Receiver
*
*
SendStage*
RcvStage*
SndRcpts
SndText
SndEnd
RcvPcpts
RcvText
RcvEnd

Abstraction

- We want to focus less implementation details and more on functionality and characteristic.
  - Implementation can change
  - Functionality derives from need
- Bad abstraction:
  - Exposing fields
  - Using drawing commands (point, line, fill…)
- Good abstraction:
  - Getters/setters
  - Shapes (rectangle, circle…)

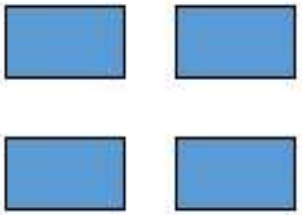
Information hiding

- We strive to hide as much details as possible, to avoid high coupling between modules.
- Bad hiding:
  - Access to unneeded methods/fields
- Better hiding
  - Using well-defined interfaces to match the needs.
- Best hiding
  - Module is operated using a defined set of command (table logic, mini-language)

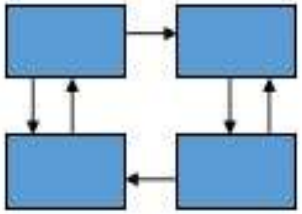
Coupling

- Coupling is a measure of the interdependence between two modules.
- Keep the modules as independent of each other as possible

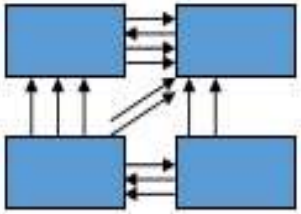
Coupling: Degree of Dependence Among Components



No dependencies



Loosely coupled-some dependencies



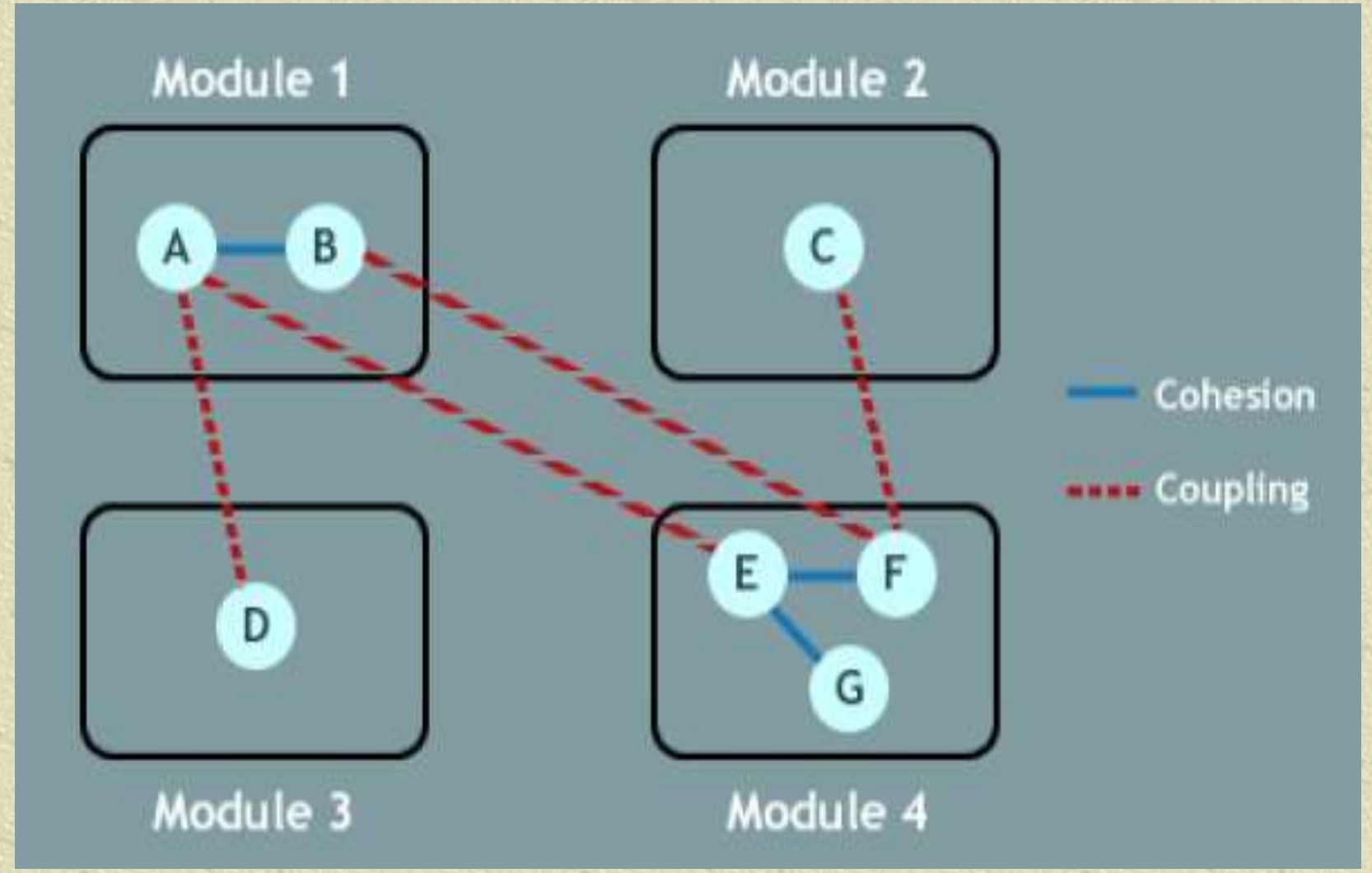
Highly coupled-many dependencies

High coupling makes modifying parts of the system difficult, e.g., modifying a component affects all the components to which the component is connected.

Cohesion

-Cohesion is the measure of the strength of functional relatedness of elements within a module.

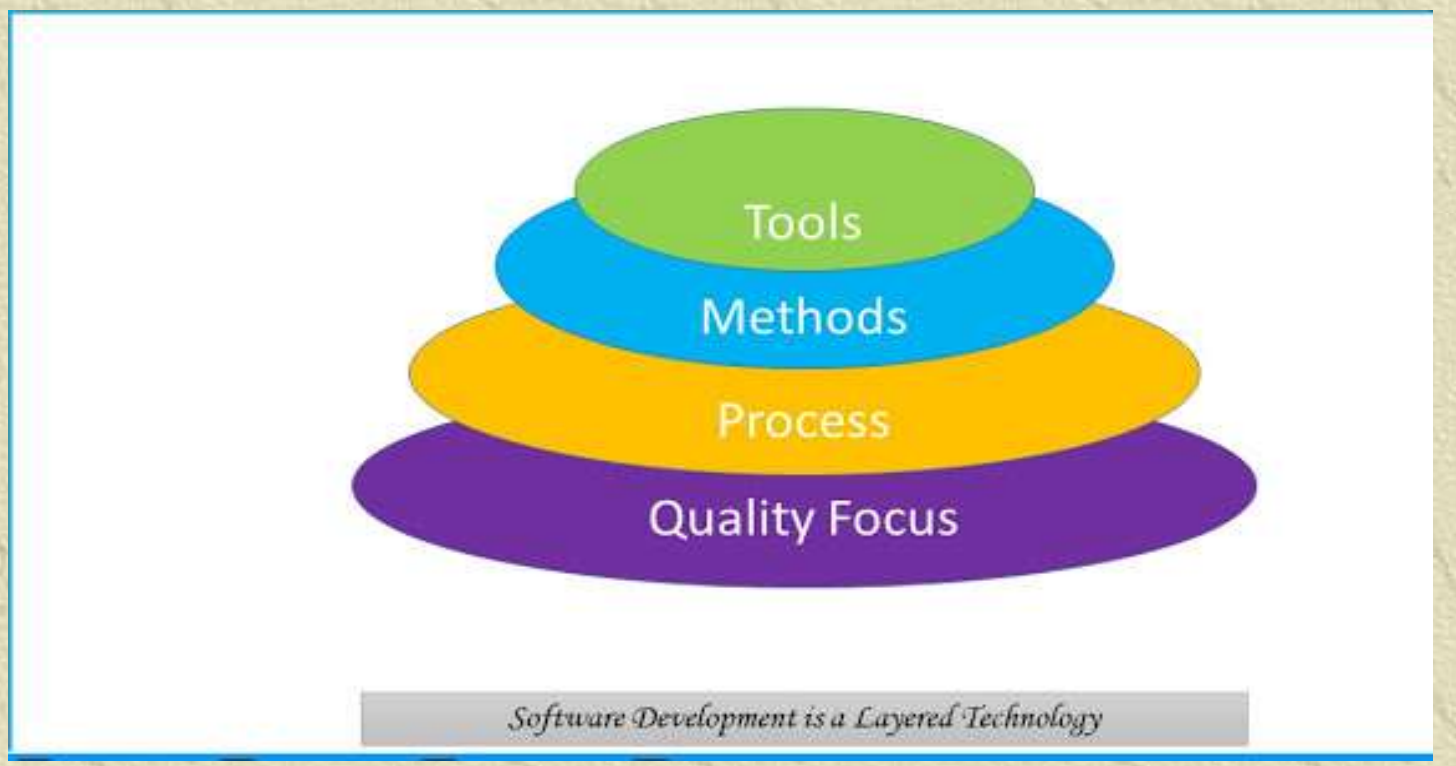
-Make each module as cohesive as possible


Module 1
Module 2
A
B
C
D
E
F
G
Module 3
Module 4
Cohesion
Coupling

Layering

- Software development is totally a layered technology.
- The layers of software development
  - Quality Focus
  - Process
  - Methods
  - Tools

The Upward Flowchart



Layering

Layering represents an ordered grouping of functionality.

-Application Specific – Top Layers

-Functionality/Business Specific – Middle Layers

-System Software – Lower Layers

Factoring

- factor the modules intelligently
- aim at a high fan-in factor
- aim at a low fan-out factor
- keep the system balanced

Modularity (1)

- High cohesion: when a change occurs, it is quickly reflected to relevant modules.
  - Bad cohesion: modules are packaged in an unrelated manner, so change is reflected slowly.
    - For example: “all utilities in the system”
  - Good cohesion: related modules are close, so it is easy to change them
    - Best: all modules responsible for carrying out a common functionality.

Modularity (2)

- Low coupling: when a change occurs, it shouldn't be reflected in unrelated modules.
- Usually a result of exposing impl. details.
  - For example: exposing the internal structure of a data structure (list, tree…) to allow iteration.
- Can be avoided by using an interface or an abstraction
  - For example: providing an *itertor* to allow iteration.
  - Even better: a *foreach* function (Perl, STL, C#)

Extensibility

- You *will* need to modify your system sometime
  - New version, requirements change, bug fixes…
- The best way to do it is by extending its abilities.
  - But for that the right module needs to be extensible
- Extension can be done by:
  - Inheritance
  - Adding elements / plug-ins
  - Mini-lang scripting, table logic…

“Virtual Machines”

- Allow reusability of systems by supplying a defined set of basic, simple operations.
  - The same basic layer can serve several applications.
  - Several systems can implement the operations set.
- Some more examples:
  - OpenGL / DirectX
  - CORBA
  - SQL / ODBC/ JDBC
  - PostScript

Qualities Of Design

- Clarity
  - **easy to understand**
  - **easy to find information**
- Extensibility
  - **easy to add new features to the design without re-organization**
- Maintainability
  - **easy to make corrective modifications**

Qualities Of Design

- Implementability
  - **straight forward to implement without re-organization**
- Reliability
  - **leads to reliable implementation**
- Efficient
  - **leads to efficient implementation**

Methodology

When

(i) a Sufficient body of guidelines have been established, documented, and lead to specific design artifacts that communicate the design

(ii) there is a broad consensus of how to apply the rules and develop the products

then the approach is called a method or a methodology

Design Methods

Design methods can be broadly classified into

- Data-Oriented
- Function-Oriented
- Object-Oriented
- Formal Methods
- Component-Oriented

Data Oriented Design

- Also known as information engineering.
- Analysis is performed on a system's data entities.
- Data Requirements are extracted.
- Data Requirements drive the program design.

Data Oriented Design

- Entities are determined for each subsystem.
- Entity inter-relationships are examined to develop the additional entities needed to support the relationships.
- The process iterates until the entity relationships can no longer be expanded.

.. Data-Oriented Design

The callable components are structured in accordance with the structure of the data.

The architecture of the system is derived from the earlier specified architecture of data, i.e., the entities and their relationships.

Can be modeled by an Entity-Relationship Diagram – ERD.

.. Data-Oriented Design

Data must be defined in detail first.

An implication of the above fact is that, when the data structure changes, the structure of the program units must also change.

.. Data Oriented Design

- is useful for systems that process lots of data
- that will be programmed using a procedural language such as COBOL.
- e.g. database and banking applications

Function-Oriented Design

- Also known as Structured Design.
- Process Decomposition is done.
- Process Requirements are extracted.
- Process Requirements drive the program design.

.. Function-Oriented Design

Fundamental principle is to partition the program into a set of steps (sub programs) that can be considered as black boxes, hiding the details of their processing.

Sub programs represent the functions to be accomplished to satisfy the requirements of the user.

This is called functional decomposition.

.. Function-Oriented Design

Partitioning is hierarchical and ordered.

The top level of the program should provide the complete solution calling sub-ordinate sub programs when necessary.

Sub programs in-turn call their sub ordinate sub programs and so on.

This repeats until the requirements of the program are satisfied by small size sub programs.

.. Function-Oriented Design

Decomposition has a focus on algorithmic considerations.

The Design of data parameters is addressed after the process modules are fully decomposed.

Avoidance of common data – Parameter passing mechanism.

Emphasis on Cohesion and Coupling.

.. Function-Oriented Design

The Specific design of the data is often widely visible.

Implication of the above fact is that, if the data structure is changed, much of the logic of the programming potentially at many places in the program must change as well.

Can be modeled using Structure charts, Data flow diagrams, flow charts etc.

Function Oriented Design

- is useful for process intensive systems
- those that will be programmed using a procedural language such as FORTRAN.

Object Oriented Design

Hybrid of data-oriented and function-oriented design.

Break the system into objects.

Objects are cohesive units of related attributes and methods.

.. Object-Oriented Design

Emphasis on Data Abstraction.

Keywords: Class, Object, Inheritance etc.

Binding the related data and methods together.

Can be modeled using Class diagrams.

(already discussed in detail in OOAD module)

.. Object Oriented Design

- useful for any system that will be programmed using an object oriented language such as C++.
- particularly well suited to developing large software systems.

Formal Methods

Use a formal language to describe a software artifact such as a specification, design, source code.

The language enables the application of formal proofs to assess the correctness of an artifact.

Formal:

in a language with mathematically defined syntax and semantics.

Example:

Notation

customer-name = title + first-name + (middle-name) + last-name +

and

title = [Mr | Miss | Ms | Mrs | Dr | Prof] [|]

first-name = {legal-character}30 either - or

order = order-no + order-date + customer-name + address + + { order-items } ( ) optional

order-items = {order-no + item-code + qty } { }

repeating

.. Component-Oriented Design

Emphasis is on Software-Reusability

Main attention points are:

- Design and discovery of components
- Ensuring that components satisfy the software requirements
- Managing components
- Mapping rules from COD to Component Programming (java, COM, C#)
- Validation using several test cases

..Component-Oriented Design

- De-facto paradigm for developing large software systems.
- enterprise scale distributed applications
- N-tier web applications.

Real-Time Systems

Systems that are required to produce intended results at (or around) a specific point on the time scale.

They are measured both on value and the temporal domains.

Not fast but predictable.

.. Real-Time Systems

A Real Time System is one in which the correctness of the system behavior depends not only on the logical result of the system, but also on the physical instant at which these results are produced.

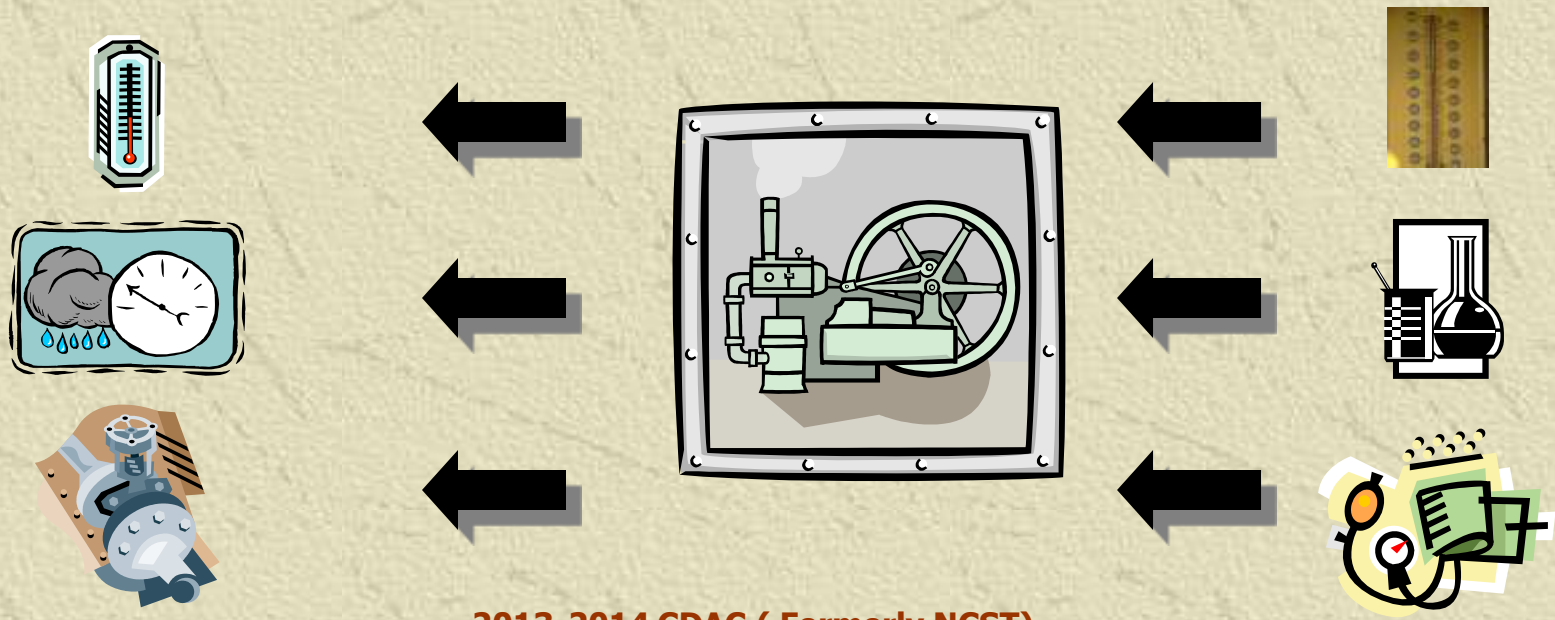
-Kopetz

Real-Time Systems- Definition

A real time system is a system that must satisfy explicitly (bounded) response time constraints or risk severe consequences, including failure, loss of life and property

Concurrent Processing

Correlated processing of two or more inputs over the same time interval



Recap