MODULE DESIGN ISSUES

From previous discussion on architectural styles:

- Want to use data abstractions to divide modules into separately understood and executing components.
- Object oriented design is a mechanism to build independent operations in each module according to these principles.

Myers and Constantine developed (in 1970s) a way to characterize degree of "separation of concerns" in module designs: Cohesion and coupling.

- Cohesion: The degree of interaction within a module
- Coupling: The degree of interaction between two modules

Goal: A design should have high cohesion and low coupling.

Note: These are interesting concepts, but one shouldn't take them too seriously - somewhat subjective.

Cohesion

Remember: Higher is considered better in a good design.

1. Coincidental cohesion - A module has coincidental cohesion if it performs multiple, unrelated actions.

Usual derivation: Clumping together unrelated functions for "convenience."

Example: A module of all "leftover" functions that are under 10 statements long:

Read next character

Compute square root of argument

Command to clear window on screen

…

Cohesion - II

2. Logical cohesion - A module has logical cohesion when it performs a series of related actions.

Examples:

- Modules to do input and output
- Modules to perform editing operations
- Modules to apply similar number conversions

3. Temporal cohesion - A module has temporal cohesion when it performs a series of actions related in time.

All functions in the module occur in the same time period, but are not necessarily functionally related.

Example:

- In a compiler, read character, create token, produce symbol table entry, as the scanner phase
- Functions to read input data and process it

(Question: aren't these more weakly related than logical cohesion?)

Cohesion-III

4. Procedural cohesion - A module has procedural cohesion if it follows a series of actions related by the sequence of steps to be followed by the product.

Example:

- All operations involving the symbol table of a compiler
- All X-windows command operations

5. Communicational cohesion - A module has communicational cohesion if it performs a series of actions related by the sequence of steps to be followed by the product and if all actions are performed on the same data.

Example:

- Functions for symbol table records involving name fields
- Functions for symbol table records involving scalar data fields

Cohesion-IV

6. Informational cohesion - A module has informational cohesion if it performs a number of actions, each with its own entry point, with independent code for each action, all performed on the same data structure.

Example:

Abstract data types are examples of this, but these are not necessarily abstract data types.

7. Functional cohesion - A module that performs exactly one action or achieves a single goal has functional cohesion.

Example:

X-Windows, Parse program

Coupling

Remember - Low coupling is considered better in a good design. Following are high to low coupling order:

5. Content coupling - Two modules are content-coupled if one directly references the content of the other.

Example: One module uses the internal data structures of another. (Definitely NOT an abstract data type.)

4. Common coupling - Two modules are common-coupled if they have access to the same global data.

Example: Most modules designed in industry.

Coupling-II

3. Control coupling - Two modules are control-coupled if one passes an element of control to the other.

Example: One module passes a switch variable that tells other module what action to take. (e.g., called modules does: If switch

then do operation 1

else do operation 2)

2. Stamp coupling - Two modules are stamp-coupled if data structures are passed as arguments, but the called module operates on only some of the data.

Example: A "user" record is passed from module A to module B, but B does not use all of the fields in the record.

Coupling-III

1. Data coupling - Two modules are data-coupled if all arguments are homogeneous data items. That is, all arguments are either simple arguments or data structures where all the fields are referenced.

In this example, a data abstraction would be passed as a simple argument to modules. Only the defining module accesses the components of the record.

Again: These are interesting goals, desirable in a

design, but their characterization if somewhat subjective, so it shouldn't be an overwhelming goal.

Design Analysis

Generally need to develop 3 components of a design:

- Object model
- Dynamic model
- Functional model

I. Object model - Defines the universe of discourse for the other two models and provides a lattice of interactions to which behavior is attached.

Objects are the things (abstract data types) that interact. The initial goal is to develop the relationships among the objects. The operations that apply to each object will occur as you develop the dynamic and functional models.

- Develop and refine the various associations among the classes you consider.
- Iterate until class design seems consistent.

Design Analysis-II

II. Dynamic model - Characterize the temporal interactions of the objects and their response to events, e.g., a dataflow diagram or a state diagram (as in the cruise control example looked at earlier).

III. Functional model - Develop functional relationships among the objects. Use cases are one way to specify such relationships.

Use Cases

Requirements analysis often stated as "finding the 'shalls' in a document."

Each 'shall' has three components:

- Action - a capability requested by the software. An action is a specific subset of the functionality that can be requested.
- Actor - a stimulus to the software system. The stimulus may be external to the system or internal. An actor requests a specific functionality.
- Qualifier - a mechanism for refining the requested actor or subject (time, which [filter], frequency [how often]).

Role - a set of responsibilities for an actor. Roles usually develop into classes.

Subject - represents the item acted upon by the action. A subject satisfies the request for an action.

Use Case - a restatement of a functional software requirement. A use case consists of an actor, action and subject.

Use case examples

1. The system shall permit a staff member to enter a staff member's name.
   - Actor - Who may add the name (e.g., operator)
   - Subject - Database containing names
   - Action - Add entry for new name
2. The system shall permit a staff member to view an entire week's entry.
   - Actor - Who may view the entry (e.g., operator)
   - [Note ambiguity here. Is this any staff member or a specific designated one.]
   - Subject - Database containing week's entry. [Note: Subject qualifier is week.]
   - Action - Display entry

Use case scenarios

A scenario is a formatted description of the steps required for the completion of a use case. It consists of the text that represents the concept of how an actor interacts with the software to achieve the desired result.

- Provides traceability from requirements to design.
- Avoid implementation details at this time.
- Provides details for system testing later.

Architectural integration

Previously: Looked at architectural styles.

- Basic designs that allow information to be passed among the components of a system: Client-server, pipes and filters, blackboard systems, …
- These provide a high-level design architecture that allows basic functionality to be easily constructed using "standard" templates.

Current issues:

- Role of standards to insure compatibility among system components
- Infrastructure components that allow applications to adhere easily to certain architectural styles.
- Emphasis today on discussing IPSEs (Integrated Project Support Environments) and the underlying infrastructure they require.
- Also look at CASE (Computer Assisted Software Engineering) tools and what has happened to them.

Language standardization

Who defines a language?:

Example is: I = 1 & 2 + 3 | 4 legal in C, and what is assigned to I?

Three ways typically to answer this:

- Read language manual (Problem: Can you find one?)
- Read language standard (Problem: Have you ever seen it?)
- Write a program to see what happens. (Easy to do!)

Most do option 3, but current compilers may not give correct answer.

Standards are a mechanism so that users and vendors of various products can build systems easily using interchangeable parts. One would like all C compilers to accept the same program. (A desirable goal, but often not achievable, as we will soon see.)

Language standards defined by national and international standards bodies: ISO, IEEE, ANSI.

Role of government in standards

- Standards in US are voluntary. -- No government enforcement of standards.
- National Institute for Standards and Technology (NIST) only provides standards through its Federal Information Processing Standards (FIPS) program that are applicable to federal agencies. NIST FIPS generally endorse commercial standards.

Standardization - in theory

Process for creating standards are relatively similar among all of these groups:

1. A vested interest in some technology meet to discuss a needed standard.
2. A working group is set up to define standard in some particular technology.
3. The working group advertises for a balloting group from among members of parent organization.
4. Members of the working group agree on features for this new standard.
5. Members of the balloting group vote on proposed standard.
6. If passed, standard sent to parent organization for approval.
7. If parent group passes standard, it is published.
8. Sometimes published as a "trial use" standard.
9. Standard effective for 5 years. All are supposed to be reviewed and revised or withdrawn at that time.

Standardization - in practice

But process is flawed:

- Contentious features often omitted to gain consensus.
- Vendors do not want major changes from current products to avoid redesign and angering current customers
- When to standardize? If too early, may not be sure of correct features; if too late, too many diverse products.
- Only vendors have a vested interest in the results, so join working group. A big "yawn" for users until standard is published, and then too late.
- Balloting group self selected by working group.
- No votes "counted" by working group and "resolved."
- Users don't care until standard approved, and then it is too late!
- Standards organizations (e.g., IEEE) make a huge profit in publishing standards documents. There is a conflict of interest in doing what is right versus raising funds. (e.g., Software engineering standards in a book that is well over $100 per copy)
- Process is slow for 1990s. Can easily take 3-5 years (or more) to create and publish a standard.

Evolution of standards

In general, standards describe effects of conforming behavior (except for Ada). Behavior of non-conforming program not specified, so any extensions to a standards conforming compiler may still be standards conforming.

Standards supposed to be reviewed every 5 years:

- FORTRAN 1966, 1977, 1990
- Ada 1983, 1995
- Not quite 5 years, but at least periodically

When to standardize?

Problem: When to standardize a language?

- If too late -- many incompatible versions -- FORTRAN in 1960s was already a de facto standard, but no two were the same.
- If too early -- no experience with language -- Ada in 1983 had no running compilers
- Just right -- Probably Pascal in 1983, although rapidly becoming a "dead" language

Other languages:

- C in 1988
- LISP in 1990 -- Way too late
- De facto standards: ML - One major implementation - SML
- Smalltalk - none
- Prolog - none
- SQL - Database access (Date?)

Architectural standards

There are very few software architectural standards - POSIX ("UNIX" kernel), OSI (data communication)

Most standards are handled as corporate and consortium standards where the "name" is owned by some entity:

- UNIX by (who owns UNIX today?)
- X-Windows by X-Windows consortium
- Motif by OSF (Are they still around?)
- Many Internet standards (by Internet Working Group?) -- TCP/IP, FTP, SMTP, TELNET
- Windows API (application program interface) by Microsoft. This is the basis for the Justice Department antitrust investigations against Microsoft.
- PCTE -- Portable Common Tool Environment by ECMA (European Computer Manufacturers Association (Note: They did original "toaster" model)
- CORBA -- Common Object Request Broker Architecture by the Object Management Group (OMG)
- Java -- by Sun?
- Active-X by Microsoft (as an alternative to Java)

Standardization summary

Industry standards allow for more rapid evolution of the standard, but loss of control of who can change the standard.

Problem -- How to get various international groups to agree on the same standard?

Major unsolved problem for today -- How to evolve the formal standardization process to be more receptive to rapid changes without allowing specific corporations to control the process?

Issues to consider:

Are standards being used effectively today? --

Do we even need them?

Who should sponsor them? -- which organizations

Who should write them? -- users or vendors

Which comes first - the underlying technology or the standard?

CASE TOOLS

CASE TOOLS -- Computer Assisted Software Engineering

Goal -- Tools to help develop software products.

Three general categories of products:

- Tools -- Support only specific tasks in the development process
- Workbenches -- Support only a few activities
- Environments -- Support a large part of the software process

Tool classifications

(From: A. Fuggetta, IEEE Computer, December, 1993)

- Editing
- Programming (e.g., compiling systems)
- Verification and validation (e.g., analysis tools, syntax checkers, symbolic executors, verifiers, test case generators, test management tools)
- Configuration management (e.g., version control and librarian functions)
- Measurement (e.g., code analyzers, execution analyzers, data visualization tools)
- Project management (e.g., cost estimation, project planning, data analysis, information handling [email, bboards, project notebooks])
- Computer supported cooperative work (CSCW) groupware tools.
- Miscellaneous (e.g., hypertext, spreadsheets)

Tool classifications - II

In general only interested in tools to help (i.e., interact with) the developer; not in the underlying infrastructure.

- Therefore, tools such as data repositories are not, but a configuration management tools that uses a repository would be.

Current interest is to create groups of interacting tools -- workbenches or environments by integrating tools together:

- Data integration -- all data in the environment is managed as a consistent whole
- Control integration -- all functions in the environment have a consistent method of enactment
- Presentation integration -- all user interaction in the environment is via a consistent interface
- Process integration -- all tools interact effectively in support of a defined process

Environments

- Environments
  - Toolkits -- Loosely integrated collections of tools
  - Language centric environments -- InterLisp, Rational's Ada environment, Smalltalk, Visual Basic
  - Integrated environment -- use of standard mechanisms to integrate tools
  - Process-centric environments (still research focus)

Integrated project support environments

General model -- Toaster model described several weeks ago:
- Database component
- Communication component
- User interface component
- Policy enforcement component
- Process management component
- Basic (operating system) component
- Framework administration component

Framework: the basic infrastructure used to build software systems. Frameworks provide:
- Registration to allow new features (e.g., tools) to be added
- Encapsulation via "wrappers" so new features from other tools can be added in a transparent manner. (e.g., Microsoft Office, Netscape, Internet Explorer can all be considered frameworks where "third party" vendors can provide "plug-ins" to add audio, video, pictures, graphics, and other features to the basic package.)
- Interoperability that permits the encapsulated tools to communicate among one another.
- Provides interactions that permit certain user tasks to be accomplished.

Sample frameworks

(Question: What has happened to these and are they still being developed?)

PCTE - Portable Common Tool Environment

Developed by European Computer Manufacturers Association around 1988-1990.
- Repository-centric framework
- NIST/ECMA toaster model developed so it could be explained what PCTE did and didn't do.
- Problems with PCTE - Fine-grained data and small objects. Too much overhead in defining access methods to data in repository. Too slow.
- Additional problem - Passing of metadata between tools (See below).
- Probably a dead issue now.

PCTE problems

Meta data problems:

- Goal: Want to register different tools within a framework so that they can interoperate.
- But: Tools need to access data within a repository.
- Access of data within repository is via a scripting language (i.e., metadata)
- Often this access is the "value added" of the tool and the metadata access is proprietary.
- So interoperability via repositories has not been solved.

Softbench by Hewlett Packard

- Based on Broadcast message server, research originally done by Stephen Reiss of Brown University.
- Each process has an address. Simply send a message to the specific address. All the processes are on a party-line listening for their own address.
- Included an encapsulator for integrating tools
- Status?
- (Similar product - Atherton Backplane)

CORBA - Common Object Request Broker Architecture

- Developed by the Object Management Group, an industry trade association
- Developed in early 1990s.
- Based on broadcast messaging to pass information among processes in an environment
- Avoids metadata problem of repository-centric solutions. Simply send message to process to request information
- CORBA part of a larger OMA - Object Management Architecture to create interoperability
- Status? - Architecture defined, but not many CORBA "backplanes" are available upon which tools can be written to interface with.

Others

Cohesion from Digital: Distributed UNIX environment using Motif, DCE (UNIX desktop environment), CORBA (Communications)

East: From SGFL, based upon PCTE, a repository-based framework, using UNIX. PCTE seemed to crumble under the weight of the data repository functions. Not sure East still being sold.

I-CASE - Integrated CASE: Grand scheme where DoD would centralize all computer development into a single large environment framework.

- Proposed as a multibillion dollar procurement in early 1990s -- Obviously, such a large sum involves lots of lawyers and legal wrangling.
- Took several years to develop within DISA (a unit of DoD)
- Main proponent of I-CASE left DoD
- Eventually Logicon won the design of I-CASE
- Now simply one of many architectures that can be used for system design

Mechanisms for integrating tools

Frameworks - Already discussed. Integrate tools.

Process management tools - somewhat immature

- Models process being defined
- Coordinated integration of other tools
- Guide personnel to enact the defined process
- Assist to determine if the defined process is followed

Process Weaver one of the more popular tools.

Maintains lists by users of tasks to accomplish.

Once a task is completed (e.g., module goes through inspection, if too many errors it gets put back on developer's list; if ok, gets put on librarian list to integrate)

Repository integration -- integrate via data

- DB provides customizable data models
- DB provides mechanisms for storing and retrieving data (i.e., corporate assets)
- DB insures integrity of software assets through authentification, versioning, configurations, and access control
- DB allows development tools to access information.