Contents

The following Help Topics are available:

For Help on Help, Press F1

About This Release

This is the beta release for a Class-Responsibility-Collaboration (CRC) Object-Oriented Design Tool, CRC Tool. This release has the basic functionality to do responsibility driven design as described in *Designing Object-Oriented Software* by Wirfs-Brock, Wilkerson and Wiener. However, it lacks refinement and a few features which would be nice to have in such a tool. For example, collaboration diagrams drawn from the specification are not supported in this release. Hierarchy diagrams are supported; but there is no way to print or output them to the design document in this release. In addition, multiple inheritance in hierarchy diagrams is not handled very well.

This release is intended to solicit input on the usefulness and usability of the tool as it now exists. No features have been crippled. Output file upwards compatibility, for files created with this release, with the production release output files is guaranteed. Assertions are enabled in this release; any assertions encountered should be written down and reported. It is recommended that a save of the data file to a new file be attempted after an assertion, since a program crash may be imminent.

This beta release will expire on May 1, 1995. This program will be offered as shareware in the production version. This program is not freeware. Tentative pricing for the shareware production version is $40.00. A corporate license is $200.00.

Contact Stan Mitchell with input regarding problems or requests for this program at:

phone:	(206)-277-6007
e-mail:	stanm@users.compumedia.com.
US mail:	Stan Mitchell 3107 Smithers Ave S Renton, WA 98055

Known problems:

- an anemic help file
- inconsistent keyboard accelerators in dialogs
- not all controls in dialogs have accelerators
- the behavior of nested subsystems is untested
- the only way to select which subsystems or classes to print is by page number
- redraw of views is not optimized
- there is no way to get from Responsibility View to System View and vice versa via the keyboard

Features to be added as time permits:

- page numbers on print out
- commands to expand and contract the hierarchical tree in the System View list
- collaboration graphs
- allow selection of classes in hierarchy diagram views
- undo delete
- exception handling
- save views and window status for a design on close
- merge input design files
- option to display WYSIWYG size cards with scrollbars in card views
- read file of nouns to create classes
- consistency check; e.g. contracts have responsibilities, contracts have clients, abstract classes don't inherit from concrete classes, etc.
- selection of classes and subsystems for printing of cards
- statistics: what classes use a given contract, etc.

- hypertext design document in HTML output format
- hypertext design document in winhelp RTF format
- drag 'n drop between all views
- add font selection, card size, line height, and print backs option to print preview toolbar
- write design to a database
- provide scripting language to extract information from design

(Comments on the prioritization of features are welcome)

Getting Started

CRC Tool consists of five views:

1. A responsibility list view
2. A subsystem-class hierarchy list view
3. A class card view
4. A subsystem card view
5. A hierarchy graph view

Each view (except for the hierarchy graph view) has a current selection. When a selection has been made, the right mouse button may be used to select a command; or a command from the menubar or toolbar may be selected.

Double-clicking on an item invokes the default action (where implemented). The Responsibility View invokes the Responsibility Property Sheet when a responsibility is double-clicked. The System View displays the selected item's view when it is double-clicked. Double-clicking in the card views is not implemented at this time.

Follow the Step-by-Step Procedures to create your design.

Glossary

Most of these definitions were taken from the book *Designing Object-Oriented Software* by Wirfs-Brock, Wilkerson and Wiener. Refer to the book for more information on each topic.

Abstract Class

Abstract classes exist merely so that behavior common to a variety of classes can be factored out into one common location, where it can be defined once and reused again and again. Abstract classes are not intended to produce instances of themselves.

Attributes

Attributes are variables defined for a responsibility.

Browser View

The Browser View is a split view which consists of the Responsibility View and System View.

Class

A class is a generic specification for an arbitrary number of similar objects. It is a template for a specific kind of object. Classes allow us to describe in one place the generic behavior of a set of objects, and then to create objects that behave in that manner when we need them.

Class Card View

A class card view is a graphical representation of an index card that holds all the pertinent information about a class, one class per card. The card holds the class name, superclasses, subclasses, responsibilities and collaborations. It also provides a brief description of the class along with a flag indicating if the class is abstract or concrete.

Class Property Sheet

A Class Property Sheet is a tabbed dialog box that provides all the necessary fields related to a class for edit. It includes the class' name, the subsystem it is assigned to, a description, an abstract flag, and any superclasses, contracts, and responsibilities associated with it.

Client

A client is the object that makes the request in a collaboration.

Collaborations

Collaborations represent requests from a client to a server in fulfillment of a client responsibility. We say that an object collaborates with another if, to fulfill a responsibility, it needs to send the other object any messages. A single collaboration flows in one direction - representing a request from the client to the server.

Concrete Classes

Concrete subclasses inherit the behavior of their abstract superclasses and add other abilities unique to their purpose. They may need to redefine the default implementations of their abstract superclasses, if any, in order to behave in a way meaningful to the application of which they are a part. These fully implemented classes create instances of themselves to do the useful work in a system.

Contract

A contract between two classes represents a list of services an instance of one class can request from an instance of the other class. The object that makes the request is the client, and the object that receives the request and thereupon provides the service is the server. A service can be either the performance of some action or the return of some information. All of the services listed in a particular contract are the responsibilities of the server for that contract.

Hierarchy

A hierarchy is an inheritance relationship between related classes. The relationships include classes, superclasses and subclasses.

Hierarchy Graph

A hierarchy graph is a view that presents a graphical representation of the inheritance relationships between related classes.

Inheritance

Inheritance is the ability of one class to define the behavior and data structure of its instances as a superset of the definition of another class or classes.

Instance

An instance of a class is an object that behaves in a manner specified by a class.

Multiple Inheritance

Multiple inheritance is the ability of a class to inherit behavior from several superclasses.

Object

An object is an instance of a class.

Protocols

Protocols are member functions; things specifically asked of a class.

Parameters

Parameters are arguments to protocols.

Private Responsibilities

Private responsibilities represent behavior a class must have, but which cannot be requested by other objects.

Responsibility

Responsibilities include two key items: the knowledge an object maintains, and the actions an object can perform. The responsibilities of an object are all the services it provides for all of the contracts it supports. A service can be either the performance of some action, or the return of some information. Express responsibilities in general terms and try to keep all of a class's responsibilities at the same conceptual level.

Responsibility Property Sheet

A Responsibility Property Sheet is a tabbed dialog box that provides all the necessary fields related to a responsibility for edit. It includes the responsibility's name, the class it is assigned to, any contracts, collaborations, protocols and attributes associated with it.

Responsibility View

A Responsibility View is a view located in the Browser which is separated from the System View by a splitter bar. The Responsibility View provides a list of the responsibilities for the system.

Server

A server is the object that receives the request in a collaboration and thereupon provides the service.

Subclass

A subclass is a class that inherits behavior from another class. A subclass usually adds its own behavior to define its own unique kind of class.

Subsystem

A subsystem is a group of classes, or groups of classes and other subsystems, that collaborate among themselves to support a set of contracts.

Subsystem Card View

A subsystem card view is a graphical representation of an index card that identifies subsystems. The cards include the subsystem name, a short description of the overall purpose, contracts required by clients external to the subsystem, and beside each contract, the delegation to the internal class or subsystem that actually supports the contract.

System View

The System View is a view located in the Browser which is separated from the Responsibility View by a splitter bar. The System View displays an outline form of the various components of a design. The components include subsystems, classes, and hierarchy graphs. Special icons reflect each of the component types. The System View provides a means of viewing the design structure, as well as modifying the structure by editing and deleting components.

Superclass

A superclass is a class from which specific behavior is inherited.

The Process

The following summary of the CRC process was taken from *Designing Object-Oriented Software* by Wirfs-Brock, Wilkerson and Wiener. Refer to the book for more information. Notes in '( )'s are related to CRC Tool.

Exploratory Phase:

1. Read and understand the specification.
2. As you follow the steps below, walk through various scenarios to explore possibilities. Record the results on design cards. (Record the results on design cards; or record on design cards and transcribe into CRC Tool; or just record in CRC Tool.)

Classes

3. Extract noun phrases from the specification and build a list.
4. Look for nouns that may be hidden (for example, by the use of the passive voice), and add them to the list.
5. Identify candidate classes from the noun phrases by applying the following guidelines:
   - Model physical objects.
   - Model conceptual entities.
   - Use a single term for each concept.
   - Be wary of the use of adjectives.
   - Model categories of objects.
   - Model external interfaces.
   - Model the values of an object's attributes.
6. Identify candidates for abstract superclasses by grouping classes that share common attributes.
7. Use categories to look for classes that may be missing.
8. Write a short statement of the purpose of the class.

Responsibilities

9. Find responsibilities using the following guidelines:
   - Recall the purpose of each class, as implied by its name and specified in the statement of purpose.
   - Extract responsibilities from the specification by looking for actions and information.
   - Identify responsibilities implied by the relationships between classes.
10. Assign responsibilities to classes using the following guidelines:
    - Evenly distribute system intelligence.
    - State responsibilities as generally as possible.
    - Keep behavior with related information.
    - Keep information about one thing in one place.
    - Share responsibilities among related classes.
11. Find additional responsibilities by looking for relationships between classes.
    - Use "is-kind-of" relationships to find inheritance relationships.
    - Use "is-analogous-to" relationships to find missing superclasses.
    - Use "is-part-of" relationships to find other missing classes.

Collaborations

12. Find and list collaborations by examining the responsibilities associated with classes. Ask:
    - With whom does this class need to collaborate to fulfill its responsibilities?
    - Who needs to make use of the responsibilities defined for this class?
13. Identify additional collaborations by looking for these relationships between classes:
    - the "is-part-of" relationship,
    - the "has-knowledge-of" relationship, and

- the "depends-upon" relationship.

14. Discard classes if no classes collaborate with them, and they collaborate with no other classes.

Analysis Phase:

Hierarchies

15. Build hierarchy graphs that illustrate the inheritance relationships between classes.
16. Identify which classes are abstract and which are concrete.
17. Draw Venn diagrams representing the responsibilities shared between classes.
18. Construct class hierarchies using the following guidelines:
    - Model a "kind-of" hierarchy.
    - Factor common responsibilities as high as possible.
    - Make sure that abstract classes do not inherit from concrete classes.
    - Eliminate classes that do not add functionality.
19. Construct the contracts defined by each class using the following guidelines:
    - Group responsibilities that are used by the same clients.
    - Maximize the cohesiveness of classes.
    - Minimize the number of contracts per class.

Subsystems

20. Draw a complete collaborations graph of your system.
21. Identify possible subsystems within your design. Look for frequent and complex collaborations. Name the subsystems.
    - Classes in a subsystem should collaborate to support a small and strongly cohesive set of responsibilities.
    - Classes within a subsystem should be strongly interdependent.
22. Simplify the collaborations *between* and *within* subsystems.
    - Minimize the number of collaborations a class has with other classes or subsystems.
    - Minimize the number of classes and subsystems to which a subsystem delegates.
    - Minimize the number of different contracts supported by a class or a subsystem.

Protocols

23. Construct the protocols for each class. Refine responsibilities into sets of signatures that maximize the usefulness of classes.
    - Use a single name for each conceptual operation, wherever it is found in the system.
    - Associate a single conceptual operation with each method name.
    - If classes fulfill the same specific responsibility, make this explicit in the inheritance hierarchy.
    - Make signatures generally useful.
    - Provide default values for as many parameters as reasonable.
24. Write a design specification for each class. (automated by CRC Tool)
25. Write a design specification for each subsystem. (automated by CRC Tool)
26. Write a design specification for each contract. (automated by CRC Tool)

Step-By-Step Procedures

- Create Classes
- Create Responsibilities
- Assign Responsibilities to Classes
- Create Hierarchy
- Re-evaluate Responsibility to Class Assignments
- Create Collaborations
- Create Hierarchy Graphs
- Create Abstract Classes
- Create Contracts
- Create Subsystems
- Assign Protocols and Attributes
- Print Cards
- Create and Print Design Document