Eric Roberts
CSCI 121

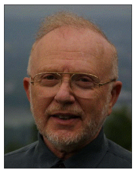
Handout #7
February 6, 2019

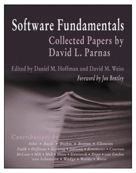
Decomposition

Decomposition

Eric Roberts
CSCS 121
February 6, 2019

David Parnas





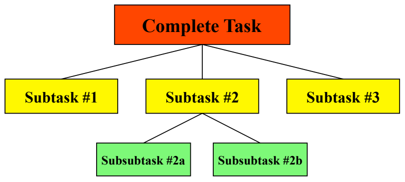
David Parnas is Professor Emeritus of Computer Science at Limerick University in Ireland, where he was director of the Software Quality Research Laboratory, and has also taught at universities in Germany, Canada, and the United States.

Parnas's most influential contribution to software engineering is his groundbreaking 1972 paper "On the criteria to be used in decomposing systems into modules," which laid the foundation for modern structured programming. This paper appears in many anthologies and is available on the web at

http://portal.acm.org/citation.cfm?id=361623

Stepwise Refinement

- The most effective way to solve a complex problem is to break it down into successively simpler subproblems.
- You start by breaking the whole task down into simpler parts.
- Some of those tasks may themselves need subdivision.
- This process is called ***stepwise refinement*** or ***decomposition***.



Criteria for Choosing a Decomposition

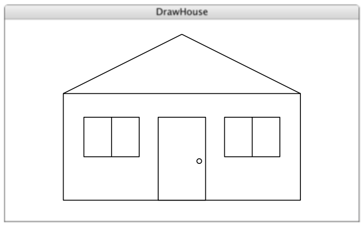
1. ***The proposed steps should be easy to explain.*** One indication that you have succeeded is being able to find simple names.
2. ***The steps should be as general as possible.*** Programming tools get reused all the time. If your methods perform general tasks, they are much easier to reuse.
3. ***The steps should make sense at the level of abstraction at which they are used.*** If you have a method that does the right job but whose name doesn't make sense in the context of the problem, it is probably worth defining a new method that calls the old one.

Passing Information between Functions

- There are several strategies you can use to pass information from one function to the functions that it calls. These include
  - ***Defining constants.*** Constants defined at the top level of a module provide a convenient way to share information that is likely to remain constant for the lifetime of a program.
  - ***Passing arguments.*** Arguments are the best mechanism for passing information that is likely to change from call to call.
  - ***Inheriting variables from enclosing functions.*** Chapter 4 introduces the idea of "inner functions" that inherit the local variables of the enclosing function. This model will have its uses, but it makes sense to postpone any discussion until later in the semester.
- There is no hard-and-fast rule for deciding which technique is best. A single application will often use all three depending on how the information is used.

Drawing a Frame House

- You can decompose the `DrawHouse` program as follows:
  - A call to `drawFrame` to draw the outline of the house
  - A call to `drawDoor` to draw the door, including the doorknob
  - Two calls to `drawWindow` to draw the windows on each side



The `DrawHouse` Program

```
# File: DrawHouse.py
"""
This program draws a simple frame house.
"""

from pgl import GWindow, GLine, GOval, GRect

# Constants

GWINDOW_WIDTH = 500
GWINDOW_HEIGHT = 300
HOUSE_WIDTH = 300
HOUSE_HEIGHT = 210
ROOF_HEIGHT = 75
DOOR_WIDTH = 60
DOOR_HEIGHT = 105
DOORKNOB_SIZE = 6
DOORKNOB_INSET_X = 5
WINDOW_WIDTH = 70
WINDOW_HEIGHT = 50
WINDOW_INSET_X = 26
WINDOW_INSET_Y = 30
```

The `DrawHouse` Program

```
def DrawHouse():
    gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT)
    houseX = (gw.getWidth() - HOUSE_WIDTH) / 2
    houseY = (gw.getHeight() - HOUSE_HEIGHT) / 2
    drawHouseAtXY(gw, houseX, houseY)

def drawHouseAtXY(gw, x, y):
    """
    Draws a simple frame house on the graphics window gw.
    The parameters x and y indicate the upper left corner
    of the bounding box that surrounds the entire house.
    """
    drawFrame(gw, x, y)
    doorX = x + (HOUSE_WIDTH - DOOR_WIDTH) / 2
    doorY = y + HOUSE_HEIGHT - DOOR_HEIGHT
    drawDoor(gw, doorX, doorY)
    leftWindowX = x + WINDOW_INSET_X
    rightWindowX = x + HOUSE_WIDTH - WINDOW_INSET_X - WINDOW_WIDTH
    windowY = y + ROOF_HEIGHT + WINDOW_INSET_Y
    drawWindow(gw, leftWindowX, windowY)
    drawWindow(gw, rightWindowX, windowY)
```

The `DrawHouse` Program

```
def drawFrame(gw, x, y):
    """
    Draws the frame for the house on the graphics window gw.
    The parameters x and y indicate the upper left corner of
    the bounding box.
    """
    roofY = y + ROOF_HEIGHT
    gw.add(GRect(x, roofY, HOUSE_WIDTH, HOUSE_HEIGHT - ROOF_HEIGHT))
    gw.add(GLine(x, roofY, x + HOUSE_WIDTH / 2, y))
    gw.add(GLine(x + HOUSE_WIDTH / 2, y, x + HOUSE_WIDTH, roofY))

def drawDoor(gw, x, y):
    """
    Draws a door (with its doorknob) on the graphics window
    gw.  The parameters x and y indicate the upper left corner
    of the door.
    """
    gw.add(GRect(x, y, DOOR_WIDTH, DOOR_HEIGHT))
    doorknobX = x + DOOR_WIDTH - DOORKNOB_INSET_X - DOORKNOB_SIZE
    doorknobY = y + DOOR_HEIGHT / 2
    gw.add(GOval(doorknobX, doorknobY, DOORKNOB_SIZE, DOORKNOB_SIZE))
```

The `DrawHouse` Program

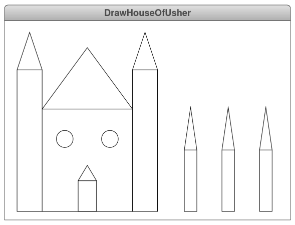
```
def drawWindow(gw, x, y):
    """
    Draws a rectangular window divided vertically into two
    panes.  The parameters x and y indicate the upper left
    corner of the window.
    """
    gw.add(GRect(x, y, WINDOW_WIDTH, WINDOW_HEIGHT))
    gw.add(GLine(x + WINDOW_WIDTH / 2, y,
                 x + WINDOW_WIDTH / 2, y + WINDOW_HEIGHT))

# Startup code

if __name__ == "__main__":
    DrawHouse()
```

Exercise: The House of Usher

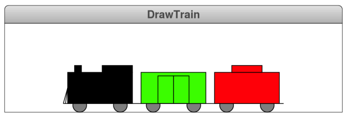
- Exercise 5 on page 108 of the text asks you to draw a stylized rendering of Edgar Allan Poe's House of Usher:



- For the next few minutes, think about how you would divide this problem into smaller parts without writing any code.

Drawing a Train

- In Chapter 10, one of the example programs is **`DrawTrain`**, which makes it possible to draw trains that look like this:



- Chapter 10 will introduce the idea of using data structures to effect the decomposition, but it is certainly possible to think about how you might draw this picture using top-down design.

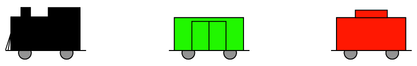
Simplifying Assumptions

- To make it easier to define the individual functions that draw a complete train, it helps to make the following assumptions:
  - The caller will want to supply the location of each car.
  - All train cars are the same size and have the same structure.
  - Engines are always black.
  - Boxcars come in many colors.
  - Cabooses are always red.
- These assumptions imply that the headers for **`drawEngine`**, **`drawBoxcar`**, and **`drawCaboose`** will look like this:

```
def drawEngine(gw, x, y):
def drawBoxcar(gw, x, y, color):
def drawCaboose(gw, x, y):
```

Looking for Common Features

- Another useful strategy in choosing a decomposition is to look for features that are shared among several different parts of a program. Such common features can be implemented by a single method.
- In the **`DrawTrain`** program, every train car has a common structure that consists of the frame for the car, the wheels on which it runs, and a connector to link it to its neighbor.
  - The engine is black and has a smokestack, cab, and cowcatcher.
  - The boxcar is colored as specified by the caller and adds doors.
  - The caboose is red and adds a cupola.



The **`DrawTrain`** Program

```
# File: DrawTrain.py
"""
This program draws a three-car train consisting on an
engine, a boxcar, and a caboose.  This implementation
is incomplete, and draws only the boxcar.
"""

from pgl import GWindow, GLine, GOval, GRect

# Constants

GWINDOW_WIDTH = 500    # Width of the graphics window
GWINDOW_HEIGHT = 200   # Height of the graphics window
CAR_WIDTH = 113        # Width of the frame of a train car
CAR_HEIGHT = 54        # Height of the frame of a train car
CAR_BASELINE = 15      # Distance of car base to the track
CONNECTOR = 6          # Width of the connector on each car
WHEEL_RADIUS = 12      # Radius of the wheels on each car
WHEEL_INSET = 24       # Distance from frame to wheel center
```

The **`DrawTrain`** Program

```
def drawBoxcar(gw, x, y, color):
    """Draws a boxcar in the specified color."""
    drawCarFrame(gw, x, y, color)
    xc = x + CONNECTOR + CAR_WIDTH / 2
    doorTop = y - CAR_BASELINE - DOOR_HEIGHT
    gw.add(GRect(xc - DOOR_WIDTH, doorTop, DOOR_WIDTH, DOOR_HEIGHT))
    gw.add(GRect(xc, doorTop, DOOR_WIDTH, DOOR_HEIGHT))

def drawCarFrame(gw, x, y, color):
    """Draws the car frame filled with the specified color."""
    x0 = x + CONNECTOR
    y0 = y - CAR_BASELINE
    top = y0 - CAR_HEIGHT
    gw.add(GLine(x, y0, x + CAR_WIDTH + 2 * CONNECTOR, y0))
    drawWheel(gw, x0 + WHEEL_INSET, y - WHEEL_RADIUS)
    drawWheel(gw, x0 + CAR_WIDTH - WHEEL_INSET, y - WHEEL_RADIUS)
    frame = GRect(x0, top, CAR_WIDTH, CAR_HEIGHT)
    frame.setFilled(True)
    frame.setFillColor(color)
    gw.add(frame)
```

The **`DrawTrain`** Program

```
def drawWheel(gw, x, y):
    """Draws a wheel centered at (x, y)."""
    wheel = GOval(x - WHEEL_RADIUS, y - WHEEL_RADIUS,
                  2 * WHEEL_RADIUS, 2 * WHEEL_RADIUS)
    wheel.setFilled(True)
    wheel.setFillColor("Gray")
    gw.add(wheel)
```