Advanced Object Oriented Programming and Design

Homework 6 – Package Design Principles and Metrics

Abraham Murciano and Daniel Klein

December 10, 2020

Question 1

a. For the package designs given in Figures 1 and 2, the following is known.

   - Classes A, B, and C do not contain virtual or pure-virtual functions.
   - Classes D, G, J, K, L, M, N, and O contain both virtual and pure-virtual functions.
   - Classes E and H contain both pure-virtual functions and functions that are neither virtual or pure virtual.
   - Classes F and I contain only virtual functions (not pure virtual).

   We are to calculate the following measures for each package, results of which are in tables 1 and 2.

C_a	Number of classes that depend on the package.
C_e	Number of classes that the package depends on.
$I = \dfrac{C_e}{c_a + C_e}$	Measure of instability.
$A = \dfrac{\#\text{abstract classes}}{\#\text{total classes}}$	Ratio of abstract classes.
$D' = \lvert A + I - 1\rvert$	
$D = \dfrac{D'}{\sqrt{2}}$	

b. The design in figure 1 violates the Stable Dependencies Principle because P_2 depends on P_3 which has a higher instability than P_2.

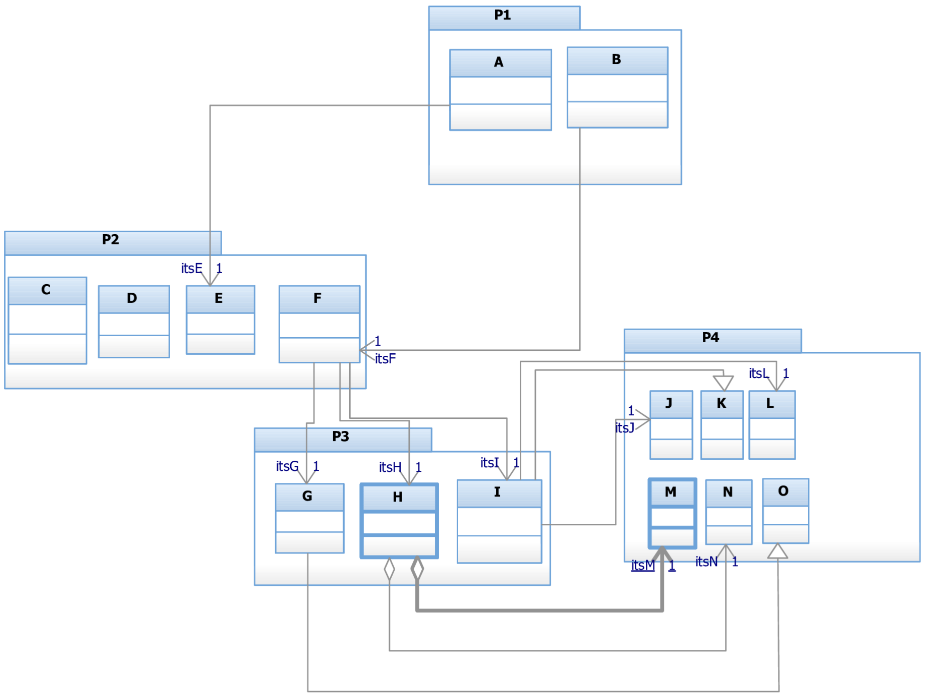

Figure 1: A package design

Class	C_a	C_e	I	A	D	D'
P_1	0	2	1	0	0	0
P_2	2	3	0.6	0.5	0.0707	0.1
P_3	2	6	0.75	$0.\dot{6}$	0.2946	$1.41\dot{6}$
P_4	3	0	0	1	0	0

Table 1: Values for part a

Class	C_a	C_e	I	A	D	D'
P_1	0	2	1	0	0	0
P_2	2	3	0.6	0.5	0.0707	0.1
P_3	1	6	$0.\dot{8}5714\dot{2}$	$0.\dot{6}$	0.3705	$1.\dot{5}2389\dot{9}$
P_4	3	0	0	1	0	0

Table 2: Values for part b

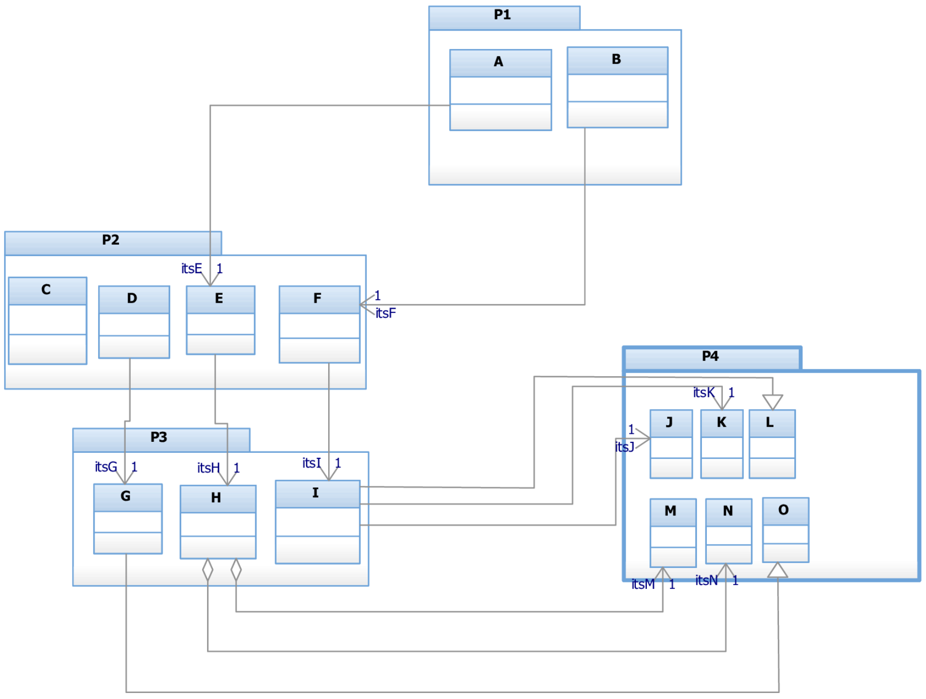

Figure 2: A package design

Question 2

a. We are to divide classes A through J into packages. The following information is given regarding the classes.

   - Client 1 reuses all the classes.
   - Client 2 reuses classes A, B and C.
   - Client 3 reuses classes A through F.
   - Classes A and D are written in Java, classes E through H are written in Python, and classes I and J are in C++.
   - A past feature update required changing classes G through J.
   - Classes G and H and classes I and J are two alternatives that implement the same pair of interfaces. A reusing project may choose to import either the first or the second couple.

   We decide to split the classes into the following packages.

$$
\begin{aligned}
P_1 &= (A, B, C) \\
P_2 &= (D, E, F) \\
P_3 &= (G, H) \\
P_4 &= (I, J)
\end{aligned}
$$

   The reason why the classes are split up into smaller packages is so that each client only uses the packages containing classes that they need. This is in accordance with the Reuse/Release Equivalence Principle.

   The reason we grouped certain classes in the same package is because they are always reused together by the clients. This follows the Common Reuse Principle.

   According to the Common Closure Principle, classes G through J should be in the same package because they are changed together. However, since they are alternative implementations of the same interface, we choose to keep them separate so that they can be reused individually in accordance to the Common Reuse Principle.

b. Assume the same dependencies between packages (ignoring the actual class names) as in figure 1. If classes A and B are changed, only package P_1 needs to be redistributed, and no other package needs to be changed, since nothing depends on P_1.