**Final Exam CMSE-520 06.01.2025 (90 min, 35 points)**

St. Name, Surname\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ St.Id#\_\_\_\_\_\_\_\_\_\_\_\_\_

**Mobiles and calculators are not allowed. Five cheat sheets with your own handwritings can be used**

Instructor Alexander Chefranov

**5 questions, 6 pages**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Point** | **4** | **4** | **3** | **12** | **12** | **35** |
| **Grade** |  |  |  |  |  |  |

**Task 1. (4 points)** What about are the Lehman’s laws? How many Lehman’s laws are there? State any four Lehman’s laws and give reasons for them.

Solution:

The Lehman’s laws are about software evolution. There are eight Lehman’s laws. Four Lehman’s laws are 1) Continuing change: if the system is not modified it eventually becomes useless; 2) Increasing complexity: if the system complexity is not reduced, it becomes too complex due to its changes; 3) Self-regulation: measures of products and processes produced by the software follow approximately normal distributions; 4) Conservation of organizational stability: the average effort to produce a new release is almost the same.

**Task 2. (4 points)** What are the four aging-related symptoms in the software? Briefly explain them.

Solution:

1. Pollution: there are many modules not used in the implementation of the functional requirements
2. Embedded knowledge: documentation does not provide enough information about the software
3. Poor lexicon: functionality of the components is not related to their identifiers
4. Coupling: components have high coupling

**Task 3. (3 points)** What three steps are followed in the partial reengineering approach?

Solution:

1. Define a part for reengineering and the rest not to be changed
2. Reengineer the part to be changed
3. Integrate the modified and not changed parts

**Task 4. (12 points)** For the directed SLO graph



the distance matrix is



Calculate distances between 1) SLO3 and SLO0 (it is 3); 2) SLO5 and SLO8 (it is also 3). Provide details of your calculations, specify respective paths, give explanations.

**Hint**: The distance between two nodes is the number of edges in the shortest path connecting them.

Solution:

1. The only way to come to SLO0 in one step is from SLO4

SLO4 can be reached in one step from SLO1, SLO2, SLO5, SLO8, and SLO9

SLO8 and SLO9 can be reached in one step from SLO3

Hence, two shortest paths from SLO3 to SLO0 are:

P1: SLO3=>SLO8=>SLO4=>SLO0

P2: SLO3=>SLO9=>SLO4=>SLO0

both having length three, and, thus, the distance between SLO3 and SLO0 is 3.

1. SLO8 can be reached in one step from SLO3 and SLO7.

SLO3 can be reached in one step from SLO2, SLO4, and SLO7, out of which only SLO4 is reachable in one step from SLO5. Hence, the shortest path from SLO5 to SLO8 is SLO5=>SLO4=>SLO3=>SLO8 having length 3. Thus, the distance from SLO5 to SLO8 is 3.

**Task 5. (12 points)** For the code below

|  |  |
| --- | --- |
|  |  |

the distance matrix is

**

Calculate distances between 1) aA1 and mA2 (0.67); 2) mB3 and mB1 (0.86). Provide details of your calculations, explain them.

**Hints**:

***Cohesion Metrics*** Simon et al., 2001, *Metrics Based Refactoring*. have introduced the concept of a *distance based* metric to express design cohesion, where cohesion refers to the degree to which module components belong together. The distance-based metric is explained in what follows. Let *B* be a set of considered properties for a special *similarity viewpoint*. Also, let *x* and *y* denote two entities (e.g., methods and attributes) of a “module” (e.g., a class) for which we are interested in finding its cohesion. Then, the *distance* between *x* and *y* with respect to the considered property set *B*, denoted by *distB*(*x*, *y*) is computed as follows:

*distB*(*x*, *y*) = 1 −|*p*(*x*) ∩ *p*(*y*)|/|*p*(*x*) ∪ *p*(*y*)| (7.1)

where *p*(*x*) = {*pi* ∈ *B*| *x* possesses property *pi*}.

For a method *f*, the set of its properties, denoted by *Bf* , is given as follows:

*Bf* = {*f* ∪ all methods directly used by*f* ∪ all attributes directly used by *f* }*.* (7.2)

Similarly, for an attribute *g*, the set of its properties, denoted by *Bg*, is given as follows:

*Bg* = {*g* ∪ all methods using *g*}*.* (7.3)

For the calculation of distance between two entities, the needed *B* is given by the union of the two corresponding sets of attributes. For example, if we are interested in calculating the distance between two methods *f*1 and *f*2, we have *B* = *Bf*1 ∪ *Bf*2.

Similarly, if we are interested in the distance between a method *f* and an attribute *g*, then we have *B* = *Bf* ∪ *Bg*.

Solution: