University of Asia Pacific

Department of CSE

Semester Final Examination, Spring 2020

Name: Rashik Rahman

Reg ID: 17201012

Year: 3rd

Semester: 2nd

Course Code: CSE 321

Course Title: Software Engineering

Date: 7.11.2020

"During Examination and upload time I will not take any help from anyone. I will give my exam all by myself."

University of Asia Pacific

Admit Card

Final-Term Examination of Spring, 2020

Financial Clearance	PAID

Registration No : 17201012
Student Name : Rashik Rahman
Program : Bachelor of Science in Computer Science and Engineering

SI.NO.	COURSE CODE	COURSE TITLE	CR.HR.	EXAM. SCHEDULE
1	CSE 313	Numerical Methods	3.00	
2	CSE 314	Numerical Methods Lab	0.75	
3	CSE 315	Peripheral & Interfacing	3.00	
4	CSE 316	Peripheral & Interfacing Lab	1.50	
5	CSE 317	Computer Architecture	3.00	
6	CSE 319	Computer Networks	3.00	
7	CSE 320	Computer Networks Lab	1.50	
8	CSE 321	Software Engineering	3.00	
9	CSE 322	Software Engineering Lab	0.75	

Total Credit: 19.50

1. Examinees are not allowed to enter the examination hall after 30 minutes of commencement of examination for mid semester examinations and 60 minutes for semester final examinations.

2. No examinees shall be allowed to submit their answer scripts before 50% of the allocated time of examination has elapsed.

3. No examinees would be allowed to go to washroom within the first 60 minutes of final examinations.

4. No student will be allowed to carry any books, bags, extra paper or cellular phone or objectionable items/incriminating paper in the examination hall.
Violators will be subjects to disciplinary action.

This is a system generated Admit Card. No signature is required.

Admit Card Generation Time: 27-Oct-2020 08:43 PM

17201012

Answer to the Q.No. 3(a)

i) If a customer pays a software contractor to develop a system, the customer will have the right to reuse the developed codes as the customer owns it. The contractor is only paid for the development of the software so, though he developed it he can't reuse it as he isn't the owner of the code anymore. He ~~cannot~~ can't have the right to reuse the code but he can obtain the right from the customer who owns it.

ii) Though the software contractor develops the code but he can't ~~reuse~~ have the right to reuse the code as a basis for a generic ~~com~~ component as he isn't the owner of the code anymore cause he was paid by a client to develop it. The client is the rightfull owner of the code now so the developer can obtain right from the client to reuse it as a basis for a generic component.

iii) In cases where the provider of the reusable component is to be reimbursed, the payment mechanism has to be one that is agreed upon by both the provider and customer. If the reusable components are well identified and ~~com~~ compact, the provider can seek a percentage of the amount its worth each time a component is reused. This is not the only way to do this. The type of payment mechanism used

will ultimately depend on the nature of software development agreement or contract.

iv) Issues associated with software reuse are:

→ Creating, maintaining and using a reusable component library. ~~Det~~ Ensuring the developers can use this library can be expensive. Development process have to be adapted to ensure that the library is used.

→ Finding, understanding and adapting reusable components. Engineers must be reasonably confident to find components to reuse.

→ Increased maintenance cost

→ Lack of tool support

→ Not-invented-here syndrome.

Answer to the Q. No. 3(b)

Yes, I can use this MVC ~~pat~~ design pattern in this scenario. Java code is given below:

→ Step 1: create model:

```
public class Course{
    private int courseId;

    public int getCourseId(){
        return courseId;
    }
    public void setCourseId(int id){
        this.courseId = id;
    }
}
```

→ Step 2: create view:

```
public class CourseView{
    public void printCourseId(int courseId){
        System.out.println("Course Id: ");
        System.out.println(courseId);
    }
}
```

(9)
17201012

→Step 3 : create ~~model~~ controller:

```
public class ControllerClass{
   private Course model;
   private CourseView view;
   public ControllerClass(Course model, CourseView view){
      this.model = model;
      this.view = view
   }
  public void setCourseId(int id){
     model.setCourseId(id);
   }
  public int getCourseId(){
     return model.getCourseId();
   }
 public void updateView(){
   .view.printCourseId(model.getCourseId);
  }

}
```

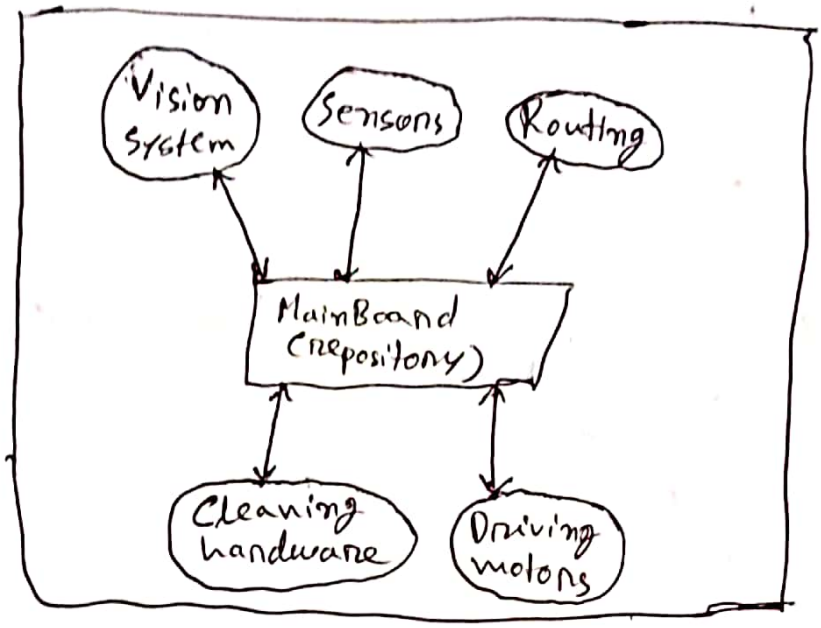
→ Step 4 : MVC patten demo

```
public class PattenClass{
    public static void main(String[] args){
        Course model = retriveCourseData();
    }

    public static Course retriveCourseData(){
        Course course = new Course();
        course.setCourseId(12);
        return course;
    }

}
```

(6)

17201012

Answer to the Q. No. 4 (a)

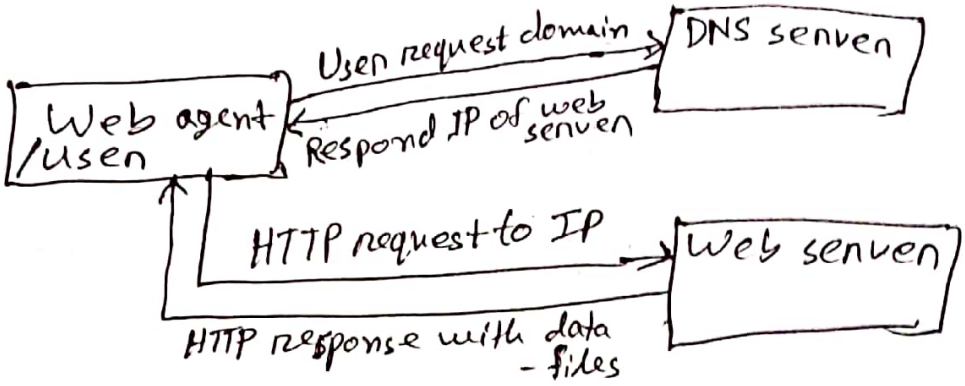


The most appropriate model for this stand alone robot floor cleaner is repository model. Cause each of the subsystems i.e. Sensors, cameras, cleaning hardware etc is placing information in the mainboard or repository for other subsystem to use. It is an effecient way to share data and the management is centralized. Thus subsystems doesn't need to store or mange data on instruction. Subsystems doesn't need to be concerned about producing data. All is managed and stored in repository.

That's why repository model would be most appropriate.

Answer to the Q. No. 4(b)

Client-server architecture is the most common architecture where the client initiates ~~connection & request & convey~~ pass a request to server and ~~as~~ server is to serve that request made by client. ~~Example: Web browsing~~ Web browsing is the most common use of this architecture.



Web browser working procedure:

→ User enters website name or file name in the browser.

→ This ~~name~~ name is sent to DNS server.

→ DNS server looksips to the corresponding IP address associated with the website name, then sents the IP address of the website's webserver as an IP respons.

→ Upon receiving the IP address browser makes a HTTP reques to that web server to access the website.

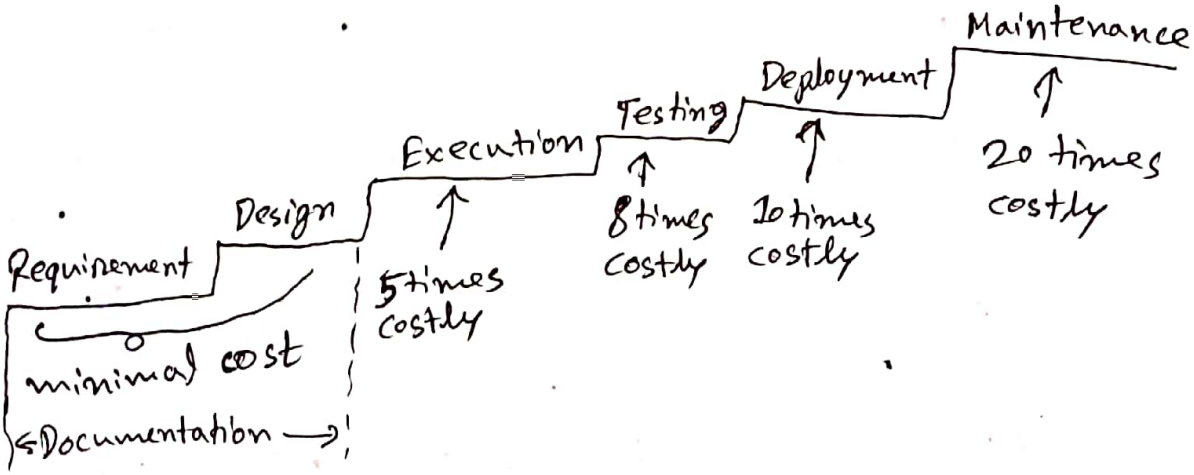
→ ~~Server then~~ Web server then sends nescessary files of the website as a HTTP response.

17201012

→ Browsen then renders the files and the website is displayed. This rendering is done with the help of DOM interpreter, CSS interpreter and JS engine.

Answer to the Q. No. 2 (a)

i) If a defect is known at the initial stage then it should be removed as it is a fact that removing defect in the early stage is most cost effective. This ~~is pictorial~~ can be seen in the following fig:



For instance if a defect is identified during requirement ~~time~~ and design then we just need to change the documentation but, if

identified during maintenance phase we not only need to fix the defect but also change our testing plans. This is why identifying defect in eany stage is ~~the~~ most cost effective.

ii)

Verification is the process of ~~identifying bug~~ checking that a software achieves its goal without any bugs. It ensures if the developen is right on not. On the other hand validation is the process to check if the product is upto the mank on nat, and checks if the product is right for the user.

So if we want to make the connect product that satisfies user needs and is right for the user we first need to make sure ~~all~~ the product is built right according to its requirement. Verification ensures product is built the right way and validation ensure developers have built the right product that ~~not~~ meets all the requirement.

So to achieve validation we'll have to make sure that verification is done properly. Basically making a successfull product that satisfies the customen is done by the means of both verification and validation. So we can say verification and validation goes hand in hand, this is the relationship between them.

(10)
17201012

Answer to the Q. No. 2(b)

When we need to update ~~the sof~~ or add a new feature to the software without impacting it's functionality we use re-engineering the re-writing.

For example if we want to update MIUI 11 to MIUI 12 with some extra features it'll be more ~~efficient~~ efficien to re-engineer then re-write. As it'll be non-sensical to re-write the whole software while we can just simply do reverse engineering and then re-structure it and do forward engineering to make the updated product. Working procedure given below:

Existing software → Reverse engineering

↙

Restructering

↙

Forwadd engineering → Re-engineered software

By doing this we also get the following advanteges:

- → Recover lost information
- → Maintenance improvement
- → Software reuse
- → Discovering unexpected errors.

Answer to the Q. No. 1(a)

ID = 17201012

size of code = 1012 KLOC

we know that when the size of the project is greated than 300kLOC then it goes under embaded category. So for this we get,

$a = 3.6$

$b = 1.2$

$c = 2.5$

$d = 0.32$

(12)

17201012

∴ Effort, E = a * (KLOC)^b

= 3.6 × (1012)^1.2 MM

= 14538.48 MM

Scheduled time (D) = c * (E)^d

= 2.5 × (14538.48)^0.32

= 53.697 Months

Average resource size = E/D = 14538.48 / 53.697 Mans

= 270.75 Mans

Productivity of software = KLOC/E = 1012 / 14538.48 KLOC/MM

= 0.06961 KLOC/MM

= 69.61 LOC/MM

Now, for organic,

a = 2.4

b = 1.05

c = 2.5

d = 0.38

Effort, $E = a*(KLOC)^b$

$= 2.9*(1012)^{1.05}$ MM

$= 3432.82$ MM

Schedule time, $D = c*(E)^d$

$= 2.5 \times (3432.82)^{0.38}$ Months

$= 55.143$ months.

Average resource size $= E/D = \frac{3432.82}{55.143}$ Mans

$= 62.2531$ Mans

Productivity of software $= KLOC/E$

$= \frac{1012}{3432.82}$ KLOC/MM

$= 0.2948$ KLOC/MM

$= 294.8$ LOC/MM

(18)
17201012

Answer to the Q.NO. 1 (b)

Possible errors that can are unlikely to discover through program inspection process are given below:

- i) Data faults: Are initialized values used? Possibility of buffer overflow? Have all constant been named?

ii) Control ~~fat~~ faults: Is a ~~code~~ condition of a condition statement correct? Are loops certained to end? Are compound statement bracketed correctly?

iii) Input/Output faults: Are all input variables used? Does all the output variables has values? Can input corruption/error happen?

iv) Interface fault: Do all function & method calls have the correct number of parameters? Do formal & actual type match? Are the parameters in right order?

v) Storage management faults:

If a linked structure is modified, have all links been correctly reassigned? Is space explicitly de-allocated after it is no longer required?

vi) Exception management faults:

Have all possible error conditions been taken into account?

—— o ——