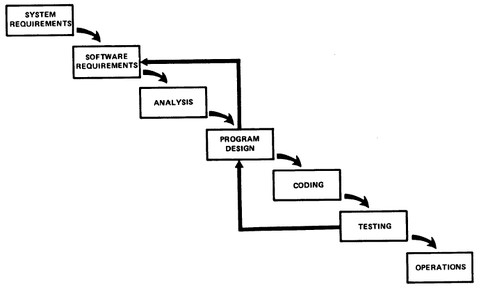
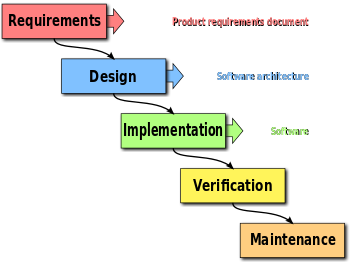
**Homework 2**

**COM S 362**

**Fall 2021**

1. The first formal description of the “waterfall model” was given by Winston W. Royce in 1970, although the name waterfall was not used until latter. The diagram on the left is often associated with his work, but in reality, Royce proposed it only as an example of what he thought was an inadequate model. His actual approach, as shown on the right, was one that allowed for going back to previous steps as flaws were encountered during testing or requirements changed after the start of design.



* 1. **(15 points)** Clearly state the similarities and differences of Royce’s waterfall model (right) with an iterative agile approach such as Scrum. 60 to 120 words.

**They both follow the same system such that establishing requirements and design happens before programming/implementation. However, in the waterfall method most of the architecture is mapped out for the rest of the project while the iterative agile the requirements keep getting redefined throughout the project lifecycle. The waterfall method also goes from stage to stage and once passed it isn’t really addressed again. However, the iterative agile leave room to go back to prior steps.**

* 1. **(15 points)** Larman describes Scrum as compatible with UP, explain why this is the case, is the model proposed by Royce also compatible with UP, why or why not? 60 to 120 words. **Scrum is compatible with UP because up has a “whatever works” attitude and any iterative method such as UP can be applied to sprints in Scrum. The waterfall method however is not compatible with UP since most of the requirements and architecture are design before programming. This doesn’t allow for many changes or iterative change**

1. **(14 points)** Fill in the blanks using the following words only once, some will not be used.

|  |  |
| --- | --- |
| actor(s) agile(s) analysis(es) ~~artifact(s)~~ association(s) attribute(s) case study(ies) concept(s) class diagram(s) design(s) domain model(s) elaboration(s) interaction diagram(s) | iterative development process(es) low representational gap(s)  pattern(s) phase(s) principle(s) risk mitigation(s) ~~responsibility(ies)~~ requirement(s) sprint(s)  UML(s)  unified process(es) ~~use case(s)~~ |

According to Larman, the most useful basic skill in object-oriented design is the assignment of responsibilities. Their assignment during the analysis phase in the iterative development process may be different than how they are assigned in the software design model, a key motivation of object oriented programming is that mapping between these two is made easy by a class diagram. An early step in capturing the requirements of a system is the creation of use cases, which are written stories describing the interactions of artifacts with the system. These interactions can be further detailed in communication and sequence diagrams collectively known as associations, which help refine our understanding of the problem and begin exploring potential design solutions. A common artifact of the software design are diagrams.

1. **(56 points)** In the following problems, chose the best matching options from the table below and complete any other listed steps.

|  |  |
| --- | --- |
| Smells | Design Principles |
| Duplicate Code  Long Function  Long Parameter List  Primitive Obsession  Large Class  Data Class | Abstraction  Encapsulation  Modularity  Hierarchy  Separation of Concerns |

* 1. The code smells Long Parameter List and Data Class are a sign of missing **encapsulation**. Correcting this will also involve the application of Hierarchy.
     + Describe a refactoring step to correct the violated principle. 1 to 2 sentences.
       - Understanding Classes and utilizing global variables if it keeps occurring in the many methods. Another option is to group similar things together and create new functions.

public class Registration {

public void addStudent(int class\_id, int age, int height, int student\_id) {

// adds a student

}

public void dropStudent(int class\_id, int age, int height, int student\_id) {

// drops a student

}

}

* 1. The code smell Duplicate Code in the following code is a sign of missing separation of concern.
     + Describe a refactoring step to remove the violation. 1 to 2 sentences.
     + This violation can be fixed by taking the similar code and extracting it into a function and replacing it with a function call.

public class Vehicle { public enum FuleType {

DIESLE, ELECTRIC, HYDROGEN, LNG, JET\_FULE

}

private FuleType fuleType;

// ...

public void addFule(double amount) { if (fuleType == FuleType.DIESLE) { // TODO: steps for adding diesel fule...

} else if (fuleType == FuleType.ELECTRIC){

// TODO: steps for adding electric fule...

} else if (fuleType == FuleType.HYDROGEN) {

// TODO: steps for adding hydrogen fule... } else if (fuleType == FuleType.LNG) { // TODO: steps for adding LNG fule...

} else if (fuleType == FuleType.JET\_FULE) {

// TODO: steps for adding jet fule...

}

// ...

}

// ...

}

**3.3.** The code smells long function and Duplicate Code in the following code are a sign of missing abstraction and a violation of modularity.

* Describe a refactoring step to remove the violation of the first principle. 1 to 2 sentences.
* Describe a refactoring step to remove the violation of the second principle (note: the refactoring steps for each principle can be performed incrementally). 1 to 2 sentences.
* The first violation can be fixed by finding likewise fragments of code and extracting them into their own functions. The second violation can be fixed by looking for similar code segments (loops that do same thing) and also extracting them into their own functions.

pubic class PairedAverage { private int[] ar1; private int[] ar2;

public void calculateEverything( ) { int ans1 = 1;

int div1 = 0;

for (int i=0; i<ar1.length; i++) {

ans1 \*= ar1[i];

}

div1 = ans1 / ar1.length;

System.out.println(“The average is " + div1);

int ans2 = 1; int div2 = 0;

For (int i=0; i<ar2.length; i++) { ans2 \*= ar2[i];

}

div2 = ans2 / ar2.length;

System.out.println("The average is " + div2); int total = ans1 + ans2;

System.out.pritnln("The total is " + total); }

}

**3.4.** The following code has the code smell Primitive Obsession.

* How does the following code violate abstraction? 1 to 2 sentences.
* How does the following code violate encapsulation? 1 to 2 sentences.
* Describe a refactoring step to improve the code. 1 to 2 sentences.

The code violates encapsulation because it doesn’t utilize private variables and so there for objects/items can be accessed or changed in any manner. It also violates abstraction because of its overuse of primitive types. The code can be fixed by having a private Event variable that holds all the attributes and then having methods that return the events.

public class EventSchedule { public int month; public int date; public String day; public int hour;

public int minute;

}