********** **Pregunta 1 - Paco - 1.5p** **********
****

Pilar: “ bws, actividades duraciones y dependencias entre actividades son las cosas que necesitamos para configurar una agenda. Se estima la duración en función del esfuerzo estimado.”

********** **Pregunta 2 - Paco - 1.5p** **********
****

Diferencia entre hacer una plan de tres niveles y de cuatro niveles (la pregunta de la práctica uno o dos, no recuerdo) Poner un ejemplo, decir si es conveniente. Podemos poner hitos en cualquier lugar? Dónde pondrías hitos? . Un par de cosas más.

Aquí está preguntando por un plan de tres niveles y cuatro niveles de un proyecto iterativo. Exactamente cómo en las prácticas. Habría que indicar cada subnivel: Fases, disciplinas, … Esta no la contesté, si alguien se anima, que nos cuente más.

Pilar: "- 4 niveles y 3 niveles se pregunta sobre el modelo up. 4 niveles : fase, iteraciones, disciplinas y actividades. 3 niveles fases, iteraciones y actividades.
- No se pueden mezclar en un proyecto 4niveles y 3 niveles porque nos basamos en cosas estándares para desarrollar sw.
Se puede poner hitos en cualquier momento, pero hay que tener en cuenta que el hito es una revisión y se toman decisiones."

********** **Pregunta 3, de Eli, 2 puntos** **********
********** ********** ********** **********

(mock y stubs)
Te dan una clase Premio

```
class Premio(){
        ClienteWebCliente cliente = new ClienteWebCleinte();
        Random generador = new Random ();
        static int probabilidad=0.1f;

        public String decirPremio(){

                if ( generarAleatorio() < probabilidad){
                        try{
                                String premio = cliente .obtenerPremio();
                                return “Ha sido premiado con “ + premio;

                        }catch(ExcepcionConcreta e){
                                return “Premiado”;
                        }
                }else{
                        return “no hay premio”;
                }
        }

        public generarAleatorio(){
                return generador.newFloat();
        }
}
```

Si tenemos este codigo de EasyMock
EasyMock mock = new Easymock(Premio.class);
Easymock.expects(“mock.generarAleatorio()”).return(“0.5f)

Terminar de escribir el mock ¿Cómo queda el código de prueba? ¿Qué se obtiene?
Respuesta despues de la revisión: este mock está mal. No se le puede pasar la clase que queremos probar.

MAL

```
public void decirPremioTest(){

        String salida;
```

```
        Premio p =new Premio();
        EasyMock mock = new Easymock(Premio.class);
        Easymock.expects(“random.generarAleatorio()”).return(“0.5f)
        Easymock.replay(mock);
        salida = p.decirPremio();

        Assert.equals(“no hay premio”, salida);
        Easymock.verify(mock);

}
```

Escribe con EasyMock pruebas para estas dos salidas:
a)“Ha sido premiado con patito de goma”
Respuesta despues de la revisión:
analizamos cuáles son nuestras dependencias: son Random y clienteWebclient. Por lo tanto de esas dos hay que hacer dos mocks. Y después hay que inyectarlo al código. Nos han puesto los atributos públicos *para facilitarnos la tarea*. Por tanto, quedaría algo como esto:

```
public void Test(){

        Random ra = new Easymock(Random);
        WebCiente we = new Easymock(Webcliente);


        Premio p =new Premio();
        Easymock.expects(ra.newFloat()).return(“0.5f);
        Easymock.expects(we.obtenerPremio()).return(“patito de goma”)
        Easymock.replay(r);
        Easymock.replay(w);          // ESTAS DOS IINEAS no las recuerdo bien
        p.generador = ra;
        p.cliente  =  we;
        salida = p.decirPremio();
        Assert.equals(“no hay premio”, salida);
        Easymock.verify(r);
        Easymock.verify(w);          // ESTAS DOS IINEAS no las recuerdo bien

}
```

Importante, en este ejercicio la gracia era probar webCliente pero ojo, al estar el random DEBEMOS hacer un mock también del random, de no ser así como fue mi caso--->0
b) excepcion por error al conectar con el servidor
Respuesta despues de la revisión:
De esta no recuerdo bien cómo era el código
Pilar escribió: “El otro era igual pero escribiendo una excepción con mocks.” Empleando .andThrow()

¿Cómo cambiaría el código si lo hubieses hecho con stub en el jUnitTest?

Respuesta despues de la revisión:

Lo más importante aquí es indicar que quitaríamos la sentencia verify, que stubs no comprueba que ...

Si no la pones compruebas el estado final , igual que hacer un stub , en lugar de comprobar el comportamiento y secuencia de llamadas que es lo que hace verify.

********** **Pregunta 4 - Eli - 1.5p** **********
****

a)Los TRES elementos indispensables para monitorizar y controlar un proyecto.
b)Pon un ejemplo concreto de cómo lo usarías con las herramientas vistas en clase
c)Si se retrasa una tarea 3 días, explica dos medidas que podrías tomar. (1.5)

a) Lo que Eli quería decir en realidad era: los tres elementos previos a la monitorización y el control de un proyecto. Esos tres elementos son:
- Linea base
- hitos
- Datos reales

Yo escribí (un poco más extenso que esto): (y me puntuó un 0.15)
- un agenda planificada (me la ha tomado como buena, como línea base)
- métricas (le hablo de holguras, EVA,...)
- tomar las decisiones adecuadas para el reajuste de agendas planificada/real

Pilar escribió:"Tienes que decir que en el project tienes la linea base, los datos reales y las holguras y definir para que es cada cosa. Luego,Se pone un ejemplo concreto Fecha de planificación es el 15 y miras a ver que pasa el 18 después de los días de retraso con las holguras y valores reales. También puedes comprobarlo con eva con bac, bwcp, bwcs con valores y explicas que comparas para ver si tienes retraso. En cualquier caso, como pongas la tabla del eva y no expliques lo anterior , está mal."

b) El ejemplo...

c) Yo le escribí: (y puntué 0.0)
- Dependiendo de la holgura de la tarea en cuestión, podríamos modificar la duración de la misma
- Por otro lado, si disponemos de unidades, podemos reasignar recursos

Ella escribió en rojo: Pero se retrasaría el proyecto? También me ha dicho que habría que especificar más para que ella entienda que sé lo de las holguras, y que en lo de las unidades, tendría que haber hablado de pedro y juan.

********** **Pregunta 5 - Miguel angel - 1.5p** **********
********** **Cobertura** **********

Pone el pom, con el plugin para cobertura configurado con branches 85% y lineas 85%, con check y clean.
Para el 50% No tendríamos que tocar nada en el pom.xml puesto que ya está el haltOnFailure
Faltaba añadir en el pom la dependencia de junit -> había que incluirlo

NOTA → No estaba el plugin del site de maven

¿Qué código añadirías para hacer los test con jUnit?
Escribe el test con jUnit para el código que recorra el 50% de las ramas:

→ Después de la corrección: En este ha llamado la atención respecto al orden de los parámetros del assert en el test

```
class Examen(){

        public void foo(int n){
                if (n<0)
                        return bar(n)
                else
                        return n;
                }

        public int bar(n){
                return -n;
        }
```

¿Qué complejidad ciclomática devolvería cobertura para la clase Examen?
(Yo he escrito: 2, porque son los caminos independientes para recorrer todas las líneas de código). Había que hacer el grafo y utilizar la fórmula.

¿Qué artefactos producen esto y qué hacen?

mvn cobertura:cobertura / no hace el check, instrumenta, genera informe
mvn site: no genera el informe de cobertura porque no está en el pom
mvn clean : limpia el directorio target
mvn install

********** **Pregunta 6 - Miguel angel - 1.5p** **********
********** **TDD** **********

Escribe el código y los artefactos que se generarían para crear 2 bibliotecas de 2D. Geometria.distancia(Point p1, Point p2) y Geometria.modulo(Vector v)

Cosas que me han llamado la atención:
- Ponía en la corrección en rojo que nos faltaba : “luz roja”
- Teníamos que percibir que distancia y módulo eran el mismo código y por tanto, al finalizar, refactorizar y convertirlo en un solo método
- Había que hablar de la refactorización “code smells”

Punto 1 y 2 SUPERimportante.