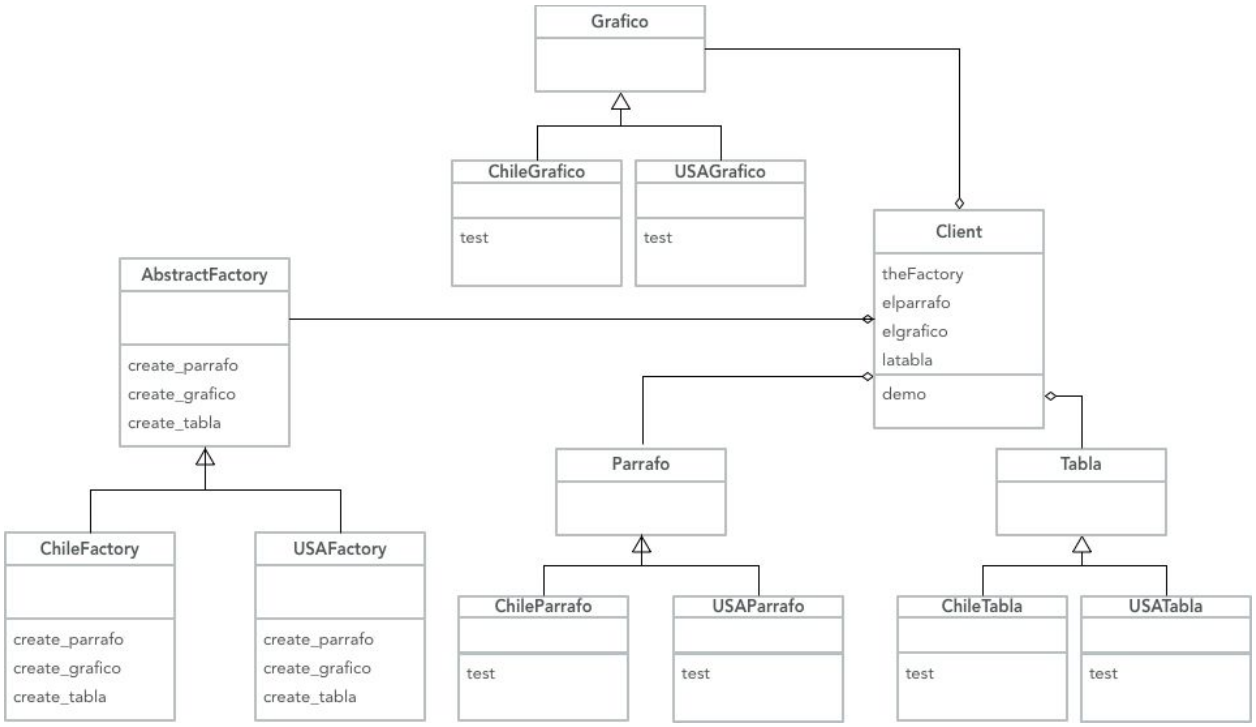
Pregunta 1

a) El problema requiere manejar familias de objetos que tienen características distintivas. Muy similar al clásico problema de manejar diferentes versiones de interfaz de usuario y por lo tanto el patrón más apropiado es el de fábrica abstracta.

b) Cada fábrica concreta será la encargada de producir instancias de los objetos concretos de una familia. En el caso de 2 tipos de reporte tendremos algo como lo siguiente (para simplificar supondremos que hay solo párrafos, tablas y gráficos, en el problema descrito hay 6 tipos de objetos por familia)




```
class AbstractFactory
   def create_parrafo
         puts "You should implement this method in the concrete factory"
   end
   def create_grafico
         puts "You should implement this method in the concrete factory
   end
   def create_tabla
         puts "You should implement this method in the concrete factory"
   end
end
```

```
class ChileFactory  < AbstractFactory
    def create_parrafo
           ChileParrafo.new
    end
    def create_grafico
           ChileGrafico.new
    def create_tabla
           ChileTabla.new
    end
end

class USAFactory  < AbstractFactory
def create_parrafo
           USAParrafo.new
    end
    def create_grafico
           USAGrafico.new
    def create_tabla
           USATabla.new
    end
end

Class ChileParrafo < Parrafo
  ...
  def test
    puts "parrafo chileno"
  end
Class ChileGrafico < Grafico
  ...
  def test
    puts "grafico chileno"
  end

…
end
Class ChileTabla < Tabla
…
  def test
    puts "tabla chilena"
  end

end

Class USAParrafo < Parrafo
…
  def test
    puts "parrafo gringo"
  end

end
```

```
Class USAGrafico < Grafico
…
  def test
    puts "grafico gringo"
  end

end
Class USATabla < Tabla
…
  def test
    puts "tabla gringa"
  end

end
```

c) Código que muestra el funcionamiento de la solución (muy similar a ejemplo visto en clases)

```
class Client
    attr_accessor :thefactory
    def initialize(afactory)
           @theFactory = aFactory
           @elparrafo = @theFactory.create_parrafo
           @elgrafico = @theFactory.create_grafico
           @latabla = @theFactory.create_tabla
    end
    def demo
           @elparrafo.test
           @elgrafico.test
           @latabla.test
    end
end

cliente1 = Client.new(ChileFactory.new)
cliente1.demo

parrafo chileno
grafico chileno
tabla chilena

cliente2 = Client.new(USAFactory.new)
cliente2.demo


parrafo gringo
grafico gringo
tabla gringa
```

Pregunta 2.

a) Se está hablando de las **grandes componentes**, sean estas librerías, servicios, etc. y las **asociaciones entre ellas incluyendo protocolos de comunicación**

b) Influye en la **satisfacción de requisitos no funcionales**: desempeño, escalabilidad, portabilidad, etc.

c) En una arquitectura orientada a servicios las **componentes no son librerías o módulos sino servicios**. Esto significa que están **débilmente acopladas y solo interactúan a través de intercambio de mensajes.** La arquitectura de microservicios **no requiere de una capa o middleware, los servicios son más pequeños y autocontenidos**

d) DevOps consiste en **derribar la barrera entre desarrollo y operaciones** (DevOps) de modo que **el ciclo ágil solo termina cuando el incremento está deployado testeado y en producción**. Ello permite reaccionar mucho mas rápido a las necesidades del negocio

e) Cada **capa representa un nivel de abstracción** mayor que está basada en la que está inmediatamente debajo. Un caso clásico es el de los protocolos de redes: http está sobre TCP que está sobre IP que está sobre el protocolo nativo. El desarrollador Web no necesita saber nada mas que http.

Pregunta 3.

Identificamos las clases de equivalencia

1. Nombre es alfabético
2. Nombre es no alfabético
3. Nombre es de menos de dos caracteres
4. Nombre es de 2 a 15 caracteres
5. Nombre es de más de 15 caracteres
6. tamaño menor que 1
7. tamaño entre 1 y 48
8. tamaño mayor que 48
9. tamaño que es número entero
10. tamaño que es un número no entero
11. tamaño que no es número
12. Tamaños en orden ascendente
13. Tamaños en desorden o descendente
14. No hay tamaños
15. Uno a 5 tamaños
16. Más de 5 tamaños
17. Nombre del item primero
18. Nombre del item no es el primero
19. Una coma separa cada elemento
20. Elementos sin coma separadora
21. Espacios en blanco sobrantes
22. Sin espacios en blanco sobrantes

Diseñamos ahora casos de prueba representantes de ellos. Cada caso de prueba debe especificar el valor esperado de retorno. Al menos debe haber un representante de cada clase (algunos pueden pertenecer a más de una clase)

Caso	Input	Output
1	xy,1	T
2	4AbcDefghijklmn,1,2,3 ,4,48	F
3	x, 4	T
5	Abcdefghijklmnop, 8, 12	F
6	Xyz, 0	F
7	XY,47	T
8	Xy, 52, 60	F
9	Xy,1, 2, 4, 8	T
10	Xy, 2.5	F

11	XY, 2, 4, o, 7	F
12	Xyz, 4, 6, 8	T
13	XY, 2,3,4,1,6	F
14	AB	F
15	Abc, 1, 3, 5, 8, 10	T
16	Abc, 1, 3, 5, 8, 10, 12	F
17	Abc, 3, 6	T
18	1, xyz	F
19	Abc, 1, 3, 5	T
20	Abc, 1 3 5	F
21	Xy, 1,2 4	T
22	xyz,2,4,8	T

Pregunta 4.

a) Falso. Ello es verdad para las **revisiones formales de código y no para el análisis estático** que solo encuentra un porcentaje menor de problemas

b) Falso. Las **pruebas unitarias por ser confeccionadas por el desarrollador podrían sacar partido de la implementación si así se quisiera**. Pueden ser de caja negra o blanca.

c) Falso. Efectivamente **hay que construir stubs pero en la opción contraria hay que construir drivers**. Son por lo tanto bastante similares en cuanto a la cantidad de pega.

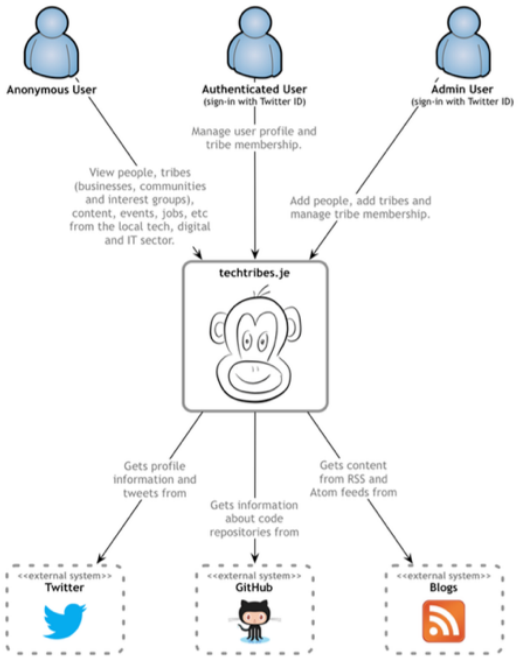
d) Verdadero. Es una **prueba con usuarios que utilizan el sistema en condiciones reales**. Sin embargo el objetivo aquí no es encontrar problemas sino seleccionar la mejor opción para el negocio

e) Falso. **Un test de regresión se hace para asegurar que los cambios hechos en una unidad no han afectado al resto**. Esto no necesariamente tiene que ver con integración.

Pregunta 5.

Hay muchas soluciones posibles, lo que se espera es algo similar a lo explicado en clases

a) Contexto: El sistema y su relación con usuarios y subsistemas o apis externas



b) Containers: Los grandes subsistemas y su comunicación