1.

a) 1. Lineas de Código (LOC)
2. Puntos de Función (FP)
3. User Story Points (USP)

Ventajas de 1 - sencillo, fácil de medir una vez construído

Desventajas de 1 - dependiente de lenguaje y estilo de programación

Ventajas de 2 y 3 - no dependen del lenguaje y del estilo, solo de la complejidad, se puede estimar en forma temprana

Desventajas de 2 y 3 - complejo y algo subjetivo

Desventaja de 2 frente a 3 - se pensó hace muchos años para software de otro tipo

b) Sirve para hacer estimaciones de tamaño en base a story points. Los participantes ponen su estimación para la historia en forma de una carta boca abajo. Se descubren al mismo tiempo y se trata de llegar a un consenso

c) Sobreestimar el esfuerzo hace que igual se use todo el tiempo y recursos disponibles lo cual introduce ineficiencia (ley de Parkinson y síndrome del estudiante). Subestimar es mucho mas dañino puesto que se generan errores y toda una dinámica de proyecto atrasado. Es mucho mas común lo segundo que lo primero.

d) Lo opuesto a economías de escala. Es decir que al aumentar el tamaño el esfuerzo no crece proporcionalmente sino mucho más. Esto se debe a que a medida que crece el proyecto hay muchas mas necesidades de comunicación y coordinación entre los desarrolladores.

2. Una tarea para el patrón observador. Los subscriptores (observadores) deben ser notificados cuando hay un nuevo episodio en el podcast (observado)
La clase Podcast debe mantener, aparte de la lista de episodios, una lista de subscriptores y debe proveer métodos para agregar un episodio, para inscribir a un sunscriptor y para eliminar a un subscriptor. El método que agrega un episodio debe encargarse de hacer las notificaciones a los subscriptores

```
class Podcast
  attr_reader :episodes, :subscribers
  def initialize()
    @episodes = []
    @subscribers = []
  end
  def add_subscriber(new_subscriber)
    subscribers << new_subscriber
  end
  def remove_subscriber(leaving_subscriber)
    subscribers.delete(leaving_subscriber)
  end
  def add_episode(new_episode)
    episodes << new_episode
    subscribers.each do |subscriber|
      subscriber.update(self)
    end
  end
end
```

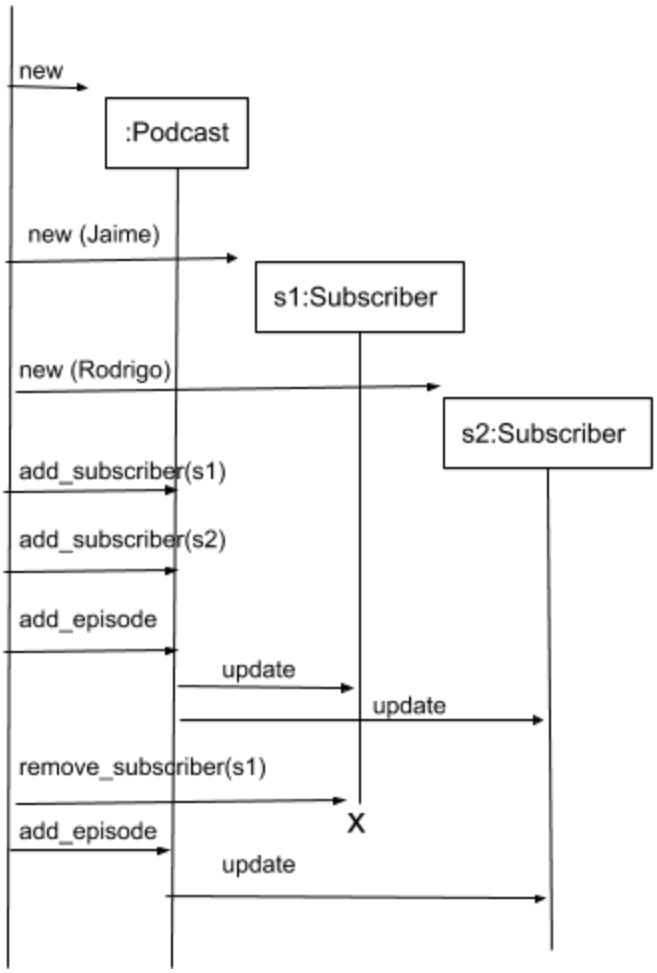
La clase Subscriber debe proveer de un método update que será invocado por el objeto Podcast. En este caso solo imprime una línea (como muestra el ejemplo)

```
class Subscriber
  attr_reader :name
  def initialize(name)
    @name = name
  end
  def update (podcast)
    puts "#{name}: there is a new episode ! "
  end
end
```

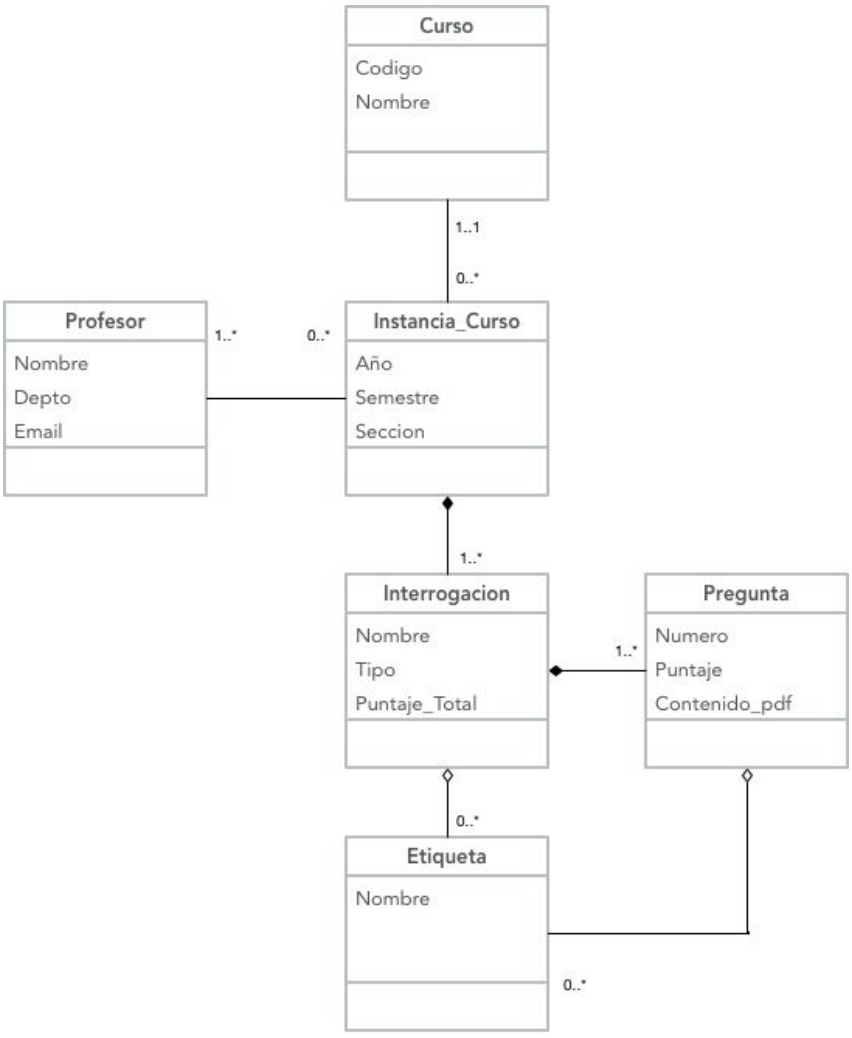
Finalmente la clase Episode no requiere nada especial

```
class Episode
  attr_reader :name
  def initialize(name)
    @name = name
  end
end
```

b) Diagrama de Secuencia



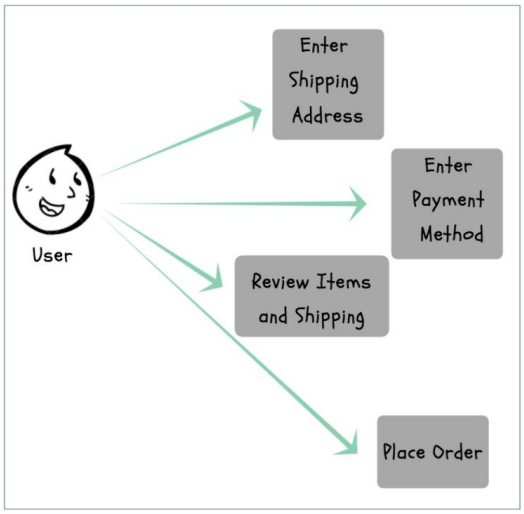
3. a) Lo importante de rescatar en el modelo es que hay instancias de un curso y que son ellas las que tienen asociadas las interrogaciones. Son las instancias también las que tienen asociados profesores. La asociación de composición se justifica porque esto está orientado a los alumnos y en este caso las preguntas son parte de una interrogación específica como también una interrogación es parte de un curso específico. Las etiquetas en cambio son compartidas y la asociación correcta es la de agregación.
Las cardinalidades elegidas indican que una instancia podría estar dada por un equipo de profesores, que una instancia de curso tiene que corresponde a uno y solo un curso, que pueden haber varias interrogaciones en una instancia y varias preguntas en una interrogación pero en ambos casos al menos una



b) Lo más sencillo es agregar un nuevo atributo a la pregunta (Solucion_pdf). Si se quiere guardar más de una solución y se quiere además poder etiquetar habría que hacer una nueva clase y asociarla a la pregunta

4. Se trata del patrón Fachada (Facade) porque lo que se busca es simplificar una interfaz. En este caso es el usuario quien directamente usará la interfaz simplificada

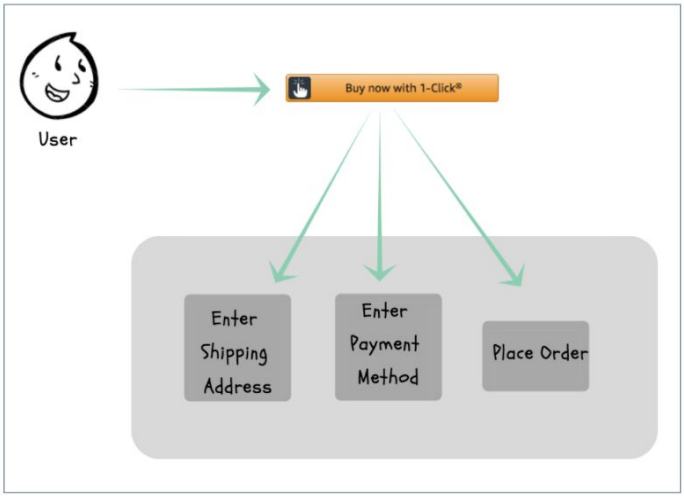
El código necesario es muy simple. Solo se debe escribir una clase OneClickCheckout que tenga un método click que llame a los métodos del checkout standard.




```
class OneClickCheckout
  attr_reader :checkout_process, :default_address,
:default_payment

  def initialize(order, default_address,
default_payment)
    @checkout_process = CheckoutProcess.new(order)
    @default_address = default_address
    @default_payment = default_payment
  end

  def click
    @checkout_process.set_shipping_address(default_address)
    @checkout_process.set_payment_method(default_payment)
    @checkout_process.place_order
  end
end
```



5. Los estados relevantes son

NL - No logueado
SI - En Sign In
SU - En Sign UP
RP - Recuperando password
LO - Logueado

Se parte en un estado NL desde donde se pasa a SI con una acción de login (puede ser un click o un selector) o a un SU con una acción de registrarse. Desde SI, si las credenciales son correctas se pasa a un estado logueado (LO) y si son incorrectas vuelve al estado NL. Desde SI puede pincharse el link "signup" o el link "forget ..." lo que nos lleva a los estados SU y RP respectivamente. Desde RP hay una transición espontánea a NL. Estando en SU se mantiene mientras los datos pedidos sean incorrrectos hasta ir finalmente a LO. Sin embargo estando en SU puede darse que el usuario ya estaba registrado (email ya existía) y en ese caso se vielve al estado SI para loguearse