1)

Para obtener una primera visión del sistema Enterprise utilizará métricas de código y buen diseño haciendo foco en el tamaño y complejidad, acoplamiento y herencia.

Como se va a tener que agregar nueva funcionalidad es importante saber cuán grande será el impacto, es decir si al empezar a realizar cambios vamos a tener que cambiar muchas cosas ya existentes o inclusive si hay riesgo de “romper” algo que está funcionando correctamente, debido a esto:

Una buena medida dentro de las métricas de acoplamiento podría ser:

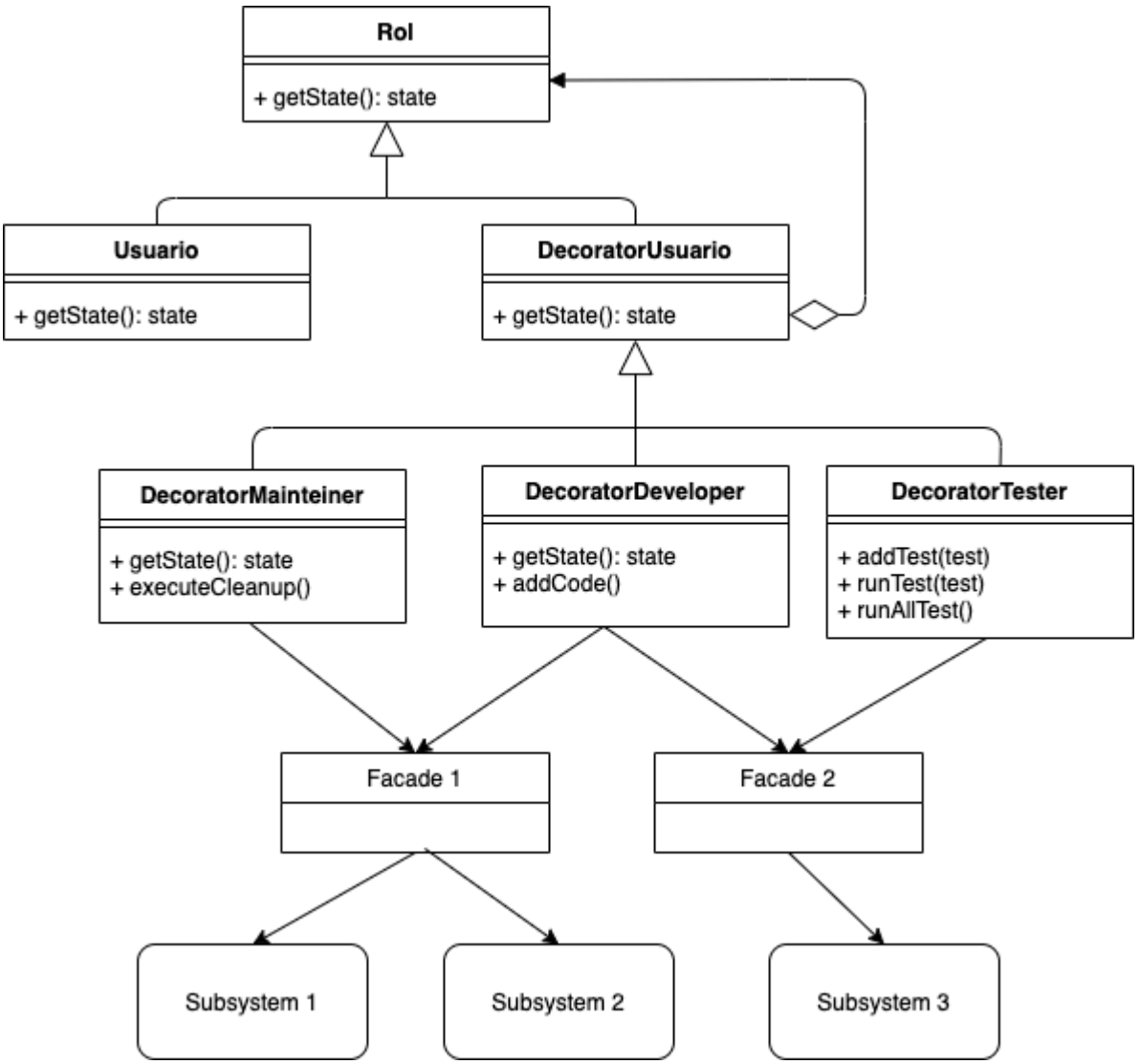
CALLS / NOM (número de invocaciones a un método dividida la cantidad de métodos)

Una buena medida dentro de las métricas de complejidad podría ser:

NOM / NOC (número de métodos de una clase dividida el número de clases)

2)

Para empezar propondría una arquitectura de **MicroKernel** debido a que hay que interactuar con sistemas externos ya implementados y además tenemos que estar abiertos a cambios en el futuro, debido a que sabemos que está es la primera versión del sistema y que el próximo año va a salir una 2da con nuevas funcionalidades y roles.
Además para aislar el nuevo sistema de los ya existentes y para no estar acoplados a los mismos recomendaría usar el patrón **Facade**.
Por otro lado como menciona que se agregaron funcionalidades para nuevos roles me inclinaría por el patrón **Decorator** ya que este permite agregar nuevas comportamientos a los objetos.
Por ej si un usuario tiene el rol TESTING y el rol DEVELOPER y se agrega un nuevo rol MAINTAINER sería fácil de sumarlo a los ya existentes.



3)

La diferencia entre un paradigma/lenguaje de programación declarativo y uno imperativo es que en el paradigma declarativo se describe el problema y sus argumentos, no el cómo resolverlo paso a paso, y el sistema va a tratar de resolverlo, en cambio en un paradigma imperativo se describen una secuencia de pasos o instrucciones a realizar, variando el estado del programa para llegar al resultado esperado.

La programación funcional pertenece al paradigma declarativo ya que describimos mediante funciones lo que queremos hacer y no damos instrucciones explícitas a ejecutar ni manejamos los estados del sistema.

Código imperativo javascript

```
const sumarImpares = (numbers) => {
    let result = 0;
    for (let i = 0; i < numbers.length; i++) {
        if(numbers[i] % 2){
            result += numbers[i];
        }
    }
    return result;
};
```

Código declarativo javascript

```
const sumarImpares = (numbers) => {
  return numbers
      .filter(n => n % 2)
      .reduce((previousVal, currentVal) => previousVal + currentVal, 0);
};
```

Podemos observar que en el paradigma declarativo no definimos como hacer la suma de los elementos ni cuales sumar, no definimos los pasos a seguir, ni declaramos un if o un for, lo contrario ocurre en el código imperativo.

En un proyecto se podrían usar ambos paradigmas pero no es recomendable mezclarlos ni que estén acoplados, podríamos usarlos en distintos paquetes o componentes.
Por ej usaría el paradigma imperativo cuando quiero tener el control total y exacto de lo que pasa, inclusive hasta el manejo de recursos.