**Pseudocódigo con explicaciones**

//Bags y contenidos

interface Contenido{

*String* nombre();

*void* renombrar(*String* *nuevoNombre*);

*List*<*Contenido*> verContenido();

}

class Bag implements *Contenido*{

*List*<*Contenido*> listaContenidos

*List*<*Hit*> versiones

*Usuario* dueño

*List*<*Usuario*> colaboradores

*List*<*Hit*> cambiosSugeridos

*List*<*Contenido*> verContenido(){

        return listaContenidos.map(contenido *->* contenido.verContenido().unificarLista())

        /\*

            Al mapear me quedaria una lista de lista de contenidos,

            por lo que unificarLista() haría que todos los elementos formen

            parte de una única lista

        \*/

    }

*boolean* validarUsuario(*Usuario* *usuario*){

        return usuario == dueño

    }

*void* sugerirCambio(*Hit* *hit*){

        cambiosSugeridos.add(hit)

    }

*void* aprobarCambio(*Hit* *hit*){

        hit.cambiarEstado(EstadoHit.APROBADO)

        this.modificar(hit)

        cambiosSugeridos.remove(hit)

    }

*void* rechazarCambio(*Hit* *hit*){

        hit.cambiarEstado(EstadoHit.RECHAZADO)

        cambiosSugeridos.remove(hit)

    }

*void* agregarColaborador(*Usuario* *usuario*){

        colaboradores.add(usuario)

    }

*void* quitarColaborador(*Usuario* *usuario*){

        colaboradores.remove(usuario)

    }

*void* agregarContenido(*Contenido* *contenido*){

        listaContenidos.add(contenido)

    }

*void* quitarContenido(*Contenido* *contenido*){

        listaContenidos.remove(contenido)

    }

*void* modificar(*Hit* *hit*){

       hit.aplicarCambiosSobre(this)

       versiones.add(hit)

    }

*Bag* verEstadoAnterior(*Hit* *hit*){

*Bag* estadoAnterior = this.clone()

        hit.deshacerCambiosSobre(estadoAnterior)

        return estadoAnterior

    }

}

class Texto implements *Contenido*{

*String* contenido

*List*<*Contenido*> verContenido(){

        return [this]

    }

}

abstract class ContenidoMultimedia implements *Contenido*{

*String* descripcion

*String* url

*List*<*Contenido*> verContenido(){

        return [this]

    }

}

class Video extends *ContenidoMultimedia*{

}

class Imagen extends *ContenidoMultimedia*{

}

//Modificaciones

interface Modificacion(){

*void* aplicar(*Bag* *bag*);

*void* deshacer(*Bag* *bag*);

}

/\* También entrarían los cambios de descripción, de urls, etc \*/

class CambiarNombre implements *Modificacion*(){

*String* viejoNombre

*String* nuevoNombre

*void* aplicar(*Bag* *bag*){

        viejoNombre = bag.nombre()

        bag.renombrar(nuevoNombre)

    }

*void* deshacer(*Bag* *bag*){

        bag.renombrar(viejoNombre)

    }

}

class AgregarContenido implements *Modificacion*(){

*Contenido* nuevoContenido

*void* aplicar(*Bag* *bag*){

        bag.agregarContenido(nuevoContenido)

    }

*void* deshacer(*Bag* *bag*){

        bag.quitarContenido(nuevoContenido)

    }

}

class QuitarContenido implements *Modificacion*(){

*Contenido* contenido

*void* aplicar(*Bag* *bag*){

        bag.quitarContenido(contenido)

    }

*void* deshacer(*Bag* *bag*){

        bag.agregarContenido(contenido)

    }

}

//Hits

class Hit{

*EstadoHit* estado = EstadoHit.PENDIENTE

*List*<Modificacion> modificaciones

*void* aplicarCambiosSobre(*Bag* bag){

        modificaciones.foreach(modificacion *->* modificacion.aplicar(bag))

    }

*void* deshacerCambiosSobre(*Bag* bag){

        modificaciones.foreach(modificacion *->* modificacion.deshacer(bag))

    }

*void* cambiarEstado(*EstadoHit* estado){

        this.estado = estado

    }

}

enum EstadoHit{

    APROBADO, RECHAZADO, PENDIENTE

}

//Usuarios

class Usuario{

*void* nuevoHitRequest(*Bag* *bag*, *Hit* *hit*){

        if(bag.validarUsuario(this)){

            bag.sugerirCambio(hit)

        }

    }

}

**Requerimientos**

1a)

unBag.agregarContenido(unVideo)

unBag.agregarContenido(unTexto)

1b)

otroBag.agregarContenido(unaImagen)

unBag.agregarContenido(otroBag)

2a)

unBag.verContenido()

2b)

unBag.agregarColaborador(juan)

unBag.agregarColaborador(pedro)

unBag.agregarColaborador(agustin)

unBag.quitarColaborador(pedro)

2c)

unCambio = new Hit(new CambiarNombre('viejo', 'nuevo'))

unBag.modificar(unCambio)

2d)

En este punto se realiza lo mismo que en el anterior, si bien no fue implementado en el pseudocódigo, se sobreentiende que para realizar una modificación debería validarse al usuario (la cual debe verificar si este es dueño o colaborador del Bag).

2e)

agustin.nuevoHitRequest(unBag, unCambio)

2f)

unBag.rechazarCambio(unCambio)

//o sino..

unBag.aprobarCambio(unCambio)

2g)

unBag.verEstadoAnterior(unCambio)

/\*

   En nuestro modelo de diseño, deberá inicializarse la lista de Hits con

   una snapshot de nuestro Bag inicial (sin contenido ni nada) por si se quiere ver el estado inicial (aunque quizá no sea necesario ya que

estaría vacío).

    También vale aclarar que no tomamos como cambio el añadir un

colaborador pero tranquilamente podría serlo (habria que añadir una clase que implemente a Modificacion).

\*/

2h)

unBag.rechazarCambio(unCambio)

unCambio.aplicarCambiosSobre(unBag)

**Diagrama de clases**

<http://www.plantuml.com/plantuml/png/dLNRRjiy3BxtKmYzSdwq_mji2vgc0LfWFQ3jTh4xsc8T3hBKwc0KA_PGUuQzs6Jv8BbfCcEvYKL-93zIzETJOq5RLuki560Cko3BNZFcVnlluQezOtXYdwBf_zlDzVdbvlfX4w2VoTYFLuhJ8vLGaf9BLYSdD5c0DOeuWoVXpNeDTK7AtAj2Otu65IkWMaGKHxE54ly6Aritw8vN_fRn8lkPPIGjwaUVYwK4sUiEep7zXnqQgS_dx4UrxefVgzgJOrG_2QnHMZCjr5UwjvfanHh2xwgBICtIOJEOznCxm2koUi0Ngr8sQoKjIk9gGgGSh5cymH6vP0tgyPGdsLULnWgYRs_RNR682nsEntzlihtTavFbqNz7ate3lnRw-cv7mfzXF9k8yHb3loPiQh1_CGuqASOT_lwb8hItBrcf11HA0rSQtwOg-vLr5UfGQSQ00wHGIY18re0W3helbWzaNFUV3CY44lqxahUJsP9TfAFLge3zVetb5duS041_sx7R0soU7TbPs9WuJc1yYekvctecUf0UFUaeh4Q3lLQiV9WcDKB2Rdb4krliQ4l6AaqWc5ypZqWzzWwV7Hfh1b-_UKEpKLYwdRyO8uOzFcOxHNtcTAL2xZxX2E0E9QbeJvHjdCTKR6GGb7nouy46EjyLdiLC8V1ykXTmj0yZ-4eS5SCNR-M9xbmvOQb6Jj3HYsfoMKE5Sbu45CPgA2qBeUzYenRjcmX7Ksfw2egHc9qMSsKGfQlZTx1BlhgzkpbRdTySixlD-cBrBJp6B-IWMW8VxSao1cUdlcjNYpy0>

**Alternativa 2h)**

En el diseño original optamos simplemente por resolver este requerimiento ejecutando el cambio desde el Hit donde no habrá una validación según el estado. Como alternativa podría existir una validación que no permita que esto pase y que existan ciertos “administradores” en el sistema que tengan acceso al Hit y puedan ejecutar un método “forzarEjecucion(Bag bag)”. De esta forma nuestro sistema será más extenso (por ende pierde **simplicidad**) ya que habrá que añadir estados a los usuarios (o bien agregar una clase administrador). Pero de esta forma estaríamos mejorando temas de **robustez** ya que no se podrán generar inconsistencias (o se reducirá la cantidad de las mismas) al tratar la información del sistema.

**Bajo la solución que propongas, si tenemos un Bag "Rock Nacional" y otro "Canciones Bizarras" el cual contiene el tema "Solo le pido a Dios":**

* **¿Puede un mismo Hit mover "Solo le pido a Dios" a "Rock Nacional"?**
* **Si es posible, contar por qué. Si no, explicar a alto nivel qué cambios sería necesario hacer a la solución para soportarlo.**

No es posible hacerlo directamente si tomamos el diseño tal y como quedó, pero gracias al polimorfismo empleado en las modificaciones podemos añadir una nueva clase “TransferirContenido” que tendrá como atributo un Bag y al aplicar el método “aplicarSobre(Bag bag)” realizamos el traspaso de contenido entre ambos.

**Code Smells**

1. Hay métodos muy extensos por lo que se pierde expresividad (Ej: crearAlbum). Podría solucionarse haciendo nuevos métodos para dividir mejor las tareas.
2. Existe alto acoplamiento entre componentes (Ej: Album y Artista). Podría solucionarse añadiendo gran parte del comportamiento en una clase Artista.
3. En vista previa podría ser mejor que tire la excepción, ya que si ningún artista contiene la canción el sistema responderá un campo nulo y el usuario quedará esperando.