

# Universidad ORT Uruguay

## Facultad de Ingeniería

Bernard Wand Polak

# Obligatorio

## Diseño de aplicaciones 2

### Integrantes

- Juan Pablo Sobral - 192247
- Chiara Di Marco - 186017

### Links

- GitHub: <https://github.com/ORT-DA2/BetterCalm-DiMarco-Sobral>

### Docentes

- Ignacio Valle
- Gabriel Piffaretti
- Nicolás Fierro

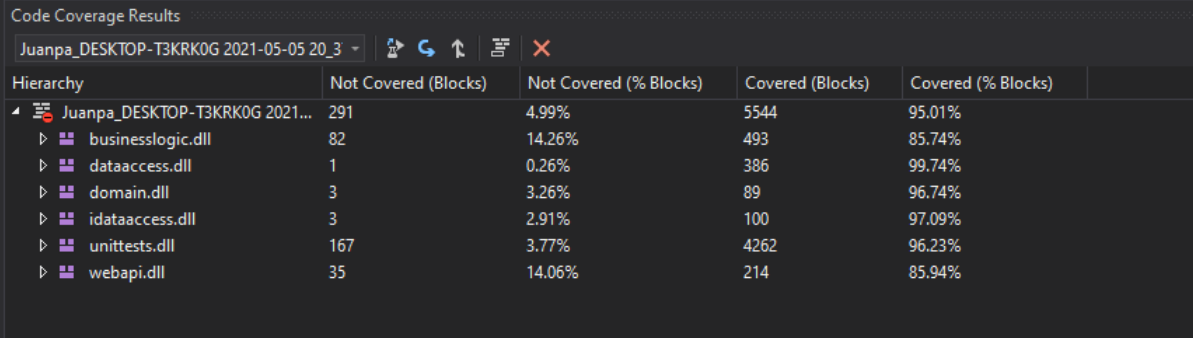
### **Aplicación de TDD:**

Se creó la aplicación teniendo en cuenta outside-in, esta consiste en verificar que el comportamiento de los objetos es el esperado. Esto se puede observar claramente en los commits realizados, donde se podrá encontrar en mayúscula las palabras:

- RED
- GREEN
- REFACTOR

RED haciendo referencia a la creación de un test unitario fallido, GREEN a la mínima implementación de código para que este pase y REFACTOR en caso necesario, para la optimización de código.

Se aplicaron los test doubles de tipo mock. Consideramos que para evitar efectos que no son deseados en nuestros test y también asegurarnos de que el comportamiento que estamos probando corresponde a la unidad de software, estos nos iban a servir.



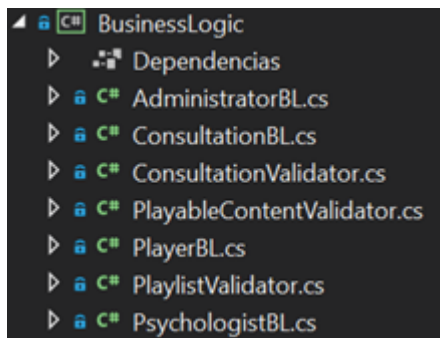
Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
Juanpa_DESKTOP-T3KRK0G 2021-05-05 20_3	291	4.99%	5544	95.01%
└─ businesslogic.dll	82	14.26%	493	85.74%
└─ dataaccess.dll	1	0.26%	386	99.74%
└─ domain.dll	3	3.26%	89	96.74%
└─ idataaccess.dll	3	2.91%	100	97.09%
└─ unittests.dll	167	3.77%	4262	96.23%
└─ webapi.dll	35	14.06%	214	85.94%

En cuanto a la cobertura de código, alcanzamos el alto grado de 95% mediante la aplicación rigurosa de TDD. Podría ser inclusive más alto pero hubo fixes que requirieron inserción a último minuto de funciones auxiliares las cuales por motivos de tiempo no se llegó a probar.

### **Justificación de Clean Code:**

#### **Nombres nemotécnicos:**

- Se intentó que los nombres sean lo más descriptivos e intencionados posible. Se evitaron las abreviaturas, prefijos, el uso de números en variables y palabras que no tengan relación con lo que se está haciendo. También se definieron los nombres de tal forma que la búsqueda de estos sea sencilla.
- Con respecto a los nombres de las clases, no se hizo uso de verbos en las mismas.



- A diferencia de este anterior, en los métodos si se utilizaron verbos.

```
5 referencias | 3/3 pasando
public void AddPlaylist (Playlist playlist)
{
    playlistValidator.ValidPlaylist(playlist);
    playlistRepository.Add(playlist);
}
```

- En todos los mencionados se cumplió con el criterio camel case, también se intentó que los nombre sean lo más cortos posible.

```
[TestMethod]
0 referencias
public void AddContentToPlaylistTest()
{
    PlayableContent auxContent = new PlayableContent
    {
        Id = 2,
        Author = "Buitres",
        Category = category,
        CategoryId = category.Id,
        Duration = 2.2,
        ContentURL = "http://cadillac-solitario.mp3",
        ImageURL = "",
        Name = "Cadillac solitario"
    };
}
```

- Tanto las properties, métodos y clases comienzan con mayúscula.

```
73 referencias
public class Administrator
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    19 referencias | 10/10 pasando
    public int Id { get; set; }
    28 referencias | 12/12 pasando
    public string Email { get; set; }

    21 referencias | 11/11 pasando
    public string Name { get; set; }

    27 referencias | 11/11 pasando
    public string Password { get; set; }

    3 referencias | 1/1 pasando
    public string Token { get; set; }
}
```

## Funciones

- Las funciones deben ser reducidas, no más de 20 líneas. Además, deben tener nombres descriptivos (como se mencionó arriba).
- Todos los métodos realizan una única cosa
- No se realizaron métodos con más de dos parámetros, la gran mayoría contiene uno solo.

```
10 referencias | 8/8 pasando
public Consultation CreateConsultation(Consultation consultation)
{
    consultationValidator.AssignPsychologist(consultation);
    consultationValidator.IdValidRangePs(consultation.Psychologist.Id);
    consultationValidator.ValidSchedule(consultation.Psychologist);
    consultationValidator.ValidAddress(consultation);
    consultationValidator.FullSchedule(consultation);
    consultationValidator.AddToSchedule(consultation.Date, consultation.Psychologist);
    consultationRepository.Add(consultation);

    return consultation;
}
```

## Comentarios

- Los comentarios no se deben encontrar a salvo que sirvan para aclarar o que aporten a la solución como una advertencia.

## Formato

- La anchura de las líneas de código no pasa los 90 caracteres
- Los espacios en blanco se utilizaron para separar conceptos
- El tamaño de las clases no excede las 500 líneas

```

[TestMethod]
✓ | 0 referencias
public void AddContentToPlaylistTest()
{
    PlayableContent auxContent = new PlayableContent
    {
        Id = 2,
        Author = "Buitres",
        Category = category,
        CategoryId = category.Id,
        Duration = 2.2,
        ContentURL = "http://cadillac-solitario.mp3",
        ImageURL = "",
        Name = "Cadillac solitario"
    };

    mock.Setup(x => x.AddContentToPlaylist(playlist.Id, 2)).
        Returns(new Playlist
        {
            Id = 1,
            Category = category,
            CategoryId = category.Id,
            Description = "Rock uruguayo",
            ImageURL = "",
            Name = "Rock uruguayo",
            Contents = new List<PlayableContent> { content, auxContent }
        });
}

```

Línea: 241    Carácter: 77

## Objetos y estructuras de datos

- Se hizo uso de interfaces para esconder la implementación (abstracción de los datos)
- No se debe violar La ley de Demeter

```

16 referencias
public interface IPlayerBL
{
    4 referencias | ✓ 2/2 pasando
    public List<Category> GetCategories();
    5 referencias | ✓ 3/3 pasando
    public List<object> GetCategoryElements(int id);
    6 referencias | ✓ 4/4 pasando
    public Playlist GetPlaylist(int id);
    6 referencias | ✓ 4/4 pasando
    public PlayableContent GetPlayableContent(int id);
    6 referencias | ✓ 4/4 pasando
    public PlayableContent AddIndependentContent(PlayableContent playableContent);
    5 referencias | ✓ 3/3 pasando
    public void AddPlaylist(Playlist playlist);
    8 referencias | ✓ 6/6 pasando
    public Playlist AddContentToPlaylist(int playlistId, int contentId);
    5 referencias | ✓ 3/3 pasando
    public void DeleteContent(int id);
    4 referencias | ✓ 2/2 pasando
    public void DeletePlaylist(int id);
}

```

```

6 referencias | 🟢 4/4 pasando
public PlayableContent GetPlayableContent(int contentId)
{
    contentValidator.IdInValidRange(contentId);











    return contentRepository.Get(contentId);
}

```

## Pruebas unitarias

- Existen 3 leyes de TDD:
  - No hay que crear código hasta que haya fallado un unit test
  - No hay que crear nunca más de una prueba que falle
  - El código creado debe ser el mínimo para que la prueba pase
- No debe realizar más de un assert por prueba
- Se debe cumplir con el principio FIRST

Aquí se puede observar cómo en un comienzo se realizan las pruebas en la fase RED y luego, una vez implementadas y corriendo, en la fase GREEN.

<b>GREEN</b> Included validation in CreateConsultation  ChiaraDim committed 8 days ago 
<b>RED</b> new tests for assignation of psychologist  ChiaraDim committed 8 days ago
<b>Added</b> new attribute to psychologist  ChiaraDim committed 8 days ago
<b>GREEN</b> Added PsychologistsController  sobraljuanpa committed 8 days ago 
<b>RED</b> Added psychologist controller tests  sobraljuanpa committed 8 days ago
<b>GREEN</b> Implemented class ConsultationBL, all tests passing  ChiaraDim committed 8 days ago 
<b>RED</b> Created and implemented class ConsultationBLTests  ChiaraDim committed 8 days ago

Una de las características que debe cumplir un test según el principio FIRST es la independencia. Esto hace referencia a que un test no puede depender de otro. Como podemos observar en la imagen inferior, no hay dependencia alguna. Otro de los aspectos que menciona este principio es que las pruebas deben correrse de forma rápida, en la imagen también se puede observar la rápida ejecución del test. Esta prueba fue ejecutada múltiples veces y siempre se obtuvo el mismo resultado, por lo que se puede afirmar que también cumple con la letra R del principio FIRST, que corresponde con la repetibilidad. Se escribieron siempre las pruebas antes de implementar el método, esto es lo que establece Timely en el principio. Un ejemplo de esto sería la imagen, donde se muestra claramente que primero se realizaron las pruebas y luego la implementación del método. Por último, la prueba fue ejecutada de forma automática.

Estos aspectos mencionados son cumplidos por todas las pruebas.