Universidad de Cádiz
Escuela Superior de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

# ARPEGOS: AUTOMATIZED ROLEPLAYING-GAME PROFILE EXTENSIBLE GENERATOR ONTOLOGY BASED SYSTEM

AUTOR: ALEJANDRO MUÑOZ DEL ÁLAMO

Puerto Real, mes ␣ año

Universidad de Cádiz
Escuela Superior de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

# ARPEGOS: AUTOMATIZED ROLEPLAYING-GAME PROFILE EXTENSIBLE GENERATOR ONTOLOGY BASED SYSTEM

DIRECTOR: D. ALBERTO GABRIEL SALGUERO HIDALGO
CODIRECTOR: D. PABLO GARCÍA SÁNCHEZ
AUTOR: ALEJANDRO MUÑOZ DEL ÁLAMO

Puerto Real, mes ␣ año

# DECLARACIÓN PERSONAL DE AUTORÍA

Alejandro Muñoz Del Álamo con DNI 77396804-X, estudiante del Grado en Ingeniería Informática en la Escuela Superior de Ingeniería de la Universidad de Cádiz, como autor de este documento academico, titulado ARPEGOS: Automatized Roleplaying-game Profile Extensible Generator Ontology based System y presentado como trabajo final de Grado.

DECLARO QUE

Es un trabajo original, que no copio ni utilizo parte de obra alguna sin mencionar de forma clara su origen, tanto en el cuerpo del texto, como en su bibliografía, y que no empleo datos de terceros sin la debida autorización, de acuerdo con la legislación vigente. Así mismo, declaro que soy plenamente consciente de que no respetar esta obligación podrá implicar la aplicación de sanciones académicas, sin prejuicio de otras actuaciones que pudieran iniciarse.

En Puerto Real, a día ␣ de mes ␣ de año

Fdo: Alejandro Muñoz Del Álamo

## *Dedicatoria*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam maximus facilisis odio vel dignissim. Integer sed tincidunt neque. Cras vitae nisi odio. Morbi in pellentesque nulla. Nam sagittis leo nec ex ultricies, eget hendrerit velit tincidunt. Phasellus fringilla varius tellus, in aliquam purus convallis eget. Donec mattis nisi turpis, vitae ultrices lectus ornare at. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis in rutrum eros. Duis ut lacinia justo. Donec fringilla velit eu sem congue cursus. Etiam porttitor, justo gravida porttitor ultrices, eros sem lacinia ex, eget bibendum enim dolor non ligula. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

***Agradecimientos***

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam maximus facilisis odio vel dignissim. Integer sed tincidunt neque. Cras vitae nisi odio. Morbi in pellentesque nulla. Nam sagittis leo nec ex ultricies, eget hendrerit velit tincidunt. Phasellus fringilla varius tellus, in aliquam purus convallis eget. Donec mattis nisi turpis, vitae ultrices lectus ornare at. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis in rutrum eros. Duis ut lacinia justo. Donec fringilla velit eu sem congue cursus. Etiam porttitor, justo gravida porttitor ultrices, eros sem lacinia ex, eget bibendum enim dolor non ligula. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

### *Resumen*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam maximus facilisis odio vel dignissim. Integer sed tincidunt neque. Cras vitae nisi odio. Morbi in pellentesque nulla. Nam sagittis leo nec ex ultricies, eget hendrerit velit tincidunt. Phasellus fringilla varius tellus, in aliquam purus convallis eget. Donec mattis nisi turpis, vitae ultrices lectus ornare at. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis in rutrum eros. Duis ut lacinia justo. Donec fringilla velit eu sem congue cursus. Etiam porttitor, justo gravida porttitor ultrices, eros sem lacinia ex, eget bibendum enim dolor non ligula. Lorem ipsum dolor sit amet, consectetur adipiscing elit.



### *Abstract*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam maximus facilisis odio vel dignissim. Integer sed tincidunt neque. Cras vitae nisi odio. Morbi in pellentesque nulla. Nam sagittis leo nec ex ultricies, eget hendrerit velit tincidunt. Phasellus fringilla varius tellus, in aliquam purus convallis eget. Donec mattis nisi turpis, vitae ultrices lectus ornare at. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis in rutrum eros. Duis ut lacinia justo. Donec fringilla velit eu sem congue cursus. Etiam porttitor, justo gravida porttitor ultrices, eros sem lacinia ex, eget bibendum enim dolor non ligula. Lorem ipsum dolor sit amet, consectetur adipiscing elit.



### *Palabras clave*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam maximus facilisis odio vel dignissim. Integer sed tincidunt neque. Cras vitae nisi odio. Morbi in pellentesque nulla. Nam sagittis leo nec ex ultricies, eget hendrerit velit tincidunt. Phasellus fringilla varius tellus, in aliquam purus convallis eget. Donec mattis nisi turpis, vitae ultrices lectus ornare at. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis in rutrum eros. Duis ut lacinia justo. Donec fringilla velit eu sem congue cursus. Etiam porttitor, justo gravida porttitor ultrices, eros sem lacinia ex, eget bibendum enim dolor non ligula. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

### Notación y formato

En la siguiente tabla se presenta un conjunto de convenios de notación de sintaxis.

| Notación establecida | |
|---|---|
| **negrita** | Título o texto destacado |
| *cursiva* | Texto en otro idioma, destacado, citas o nombres de aplicaciones |
| `monoespaciado` | Referencias a código fuente |
| subrayado | Advertencia para el lector |
| color | Enlace interno (rojo) |

# Índice general

**Índice general** **XI**

**Índice de figuras** **XV**

**Índice de cuadros** **XVII**

**I Prolegómeno** **1**

**1. Introducción** **5**
1.1. Motivación . . . . . 5
1.2. Alcance . . . . . 8
1.3. Glosario de Términos . . . . . 9
1.4. Estructuración del documento . . . . . 12

**2. Antecedentes** **15**
2.1. Estado del Arte . . . . . 15
2.2. Crítica al estado del arte . . . . . 18
2.3. Propuesta . . . . . 19
2.3.1. Plataforma . . . . . 19
2.3.2. Arquitectura . . . . . 19
2.3.3. Modelo de datos . . . . . 20
2.4. Metodología . . . . . 21
2.4.1. Metodologías de desarrollo software . . . . . 21
2.4.2. Metodologías Ágiles . . . . . 23
2.4.3. SCRUM . . . . . 24
2.5. Resumen . . . . . 27

**3. Planificación del proyecto** **29**
3.1. Planificación del proyecto . . . 29
3.1.1. Objetivo inicial . . . 29
3.1.2. División del trabajo . . . 29
3.1.3. Identificación de *sprints* y estimación de tiempos . . . 30
3.1.4. Resumen de la planificación del proyecto . . . 31
3.2. Organización . . . 33
3.2.1. Agentes involucrados . . . 33
3.2.2. Roles . . . 34
3.2.3. Recursos utilizados . . . 34
3.3. Costes . . . 35
3.3.1. Costes humanos . . . 35
3.3.2. Costes materiales . . . 36
3.3.3. Costes totales . . . 37
3.4. Evaluación y gestión de riesgos . . . 37
3.4.1. ¿Qué es un riesgo? . . . 37
3.4.2. Gestión de riesgos . . . 37
3.4.3. Identificación de riesgos . . . 38
3.4.4. Reducción de riesgos . . . 38

**II Desarrollo** **41**

**4. Requisitos del Sistema** **43**
4.1. Catálogo de requisitos . . . 43
4.1.1. Requisitos funcionales . . . 43
4.1.2. Requisitos no funcionales . . . 43
4.2. Alternativas de Solución . . . 45
4.2.1. Entorno de desarrollo integrado . . . 45
4.3. Solución propuesta . . . 47

**5. Análisis del Sistema** **49**
5.1. Modelo Conceptual . . . 49
5.1.1. Introducción . . . 49
5.1.2. UML . . . 50
5.1.3. Elementos . . . 50
5.1.4. Relaciones . . . 51
5.2. Modelo de Casos de Uso . . . 52
5.2.1. Actores . . . 52
5.2.2. Descripción de casos de uso . . . 53
5.2.3. Gestión de juegos . . . 53
5.2.4. Gestión de personajes . . . 57
5.2.5. Gestión de habilidades . . . 60
5.3. Modelo de Comportamiento . . . 61

5.3.1. Pantalla de selección . . . . . 61
5.3.2. Crear personaje . . . . . 62
5.3.3. Editar personaje . . . . . 62
5.3.4. Eliminar personaje . . . . . 63
5.3.5. Cálculo de habilidades . . . . . 63
5.4. Modelo de Interfaz de Usuario . . . . . 64

**6. Diseño del Sistema** **65**
6.1. Arquitectura del Sistema . . . . . 65
6.1.1. Arquitectura Física . . . . . 65
6.1.2. Arquitectura Lógica . . . . . 66
6.2. Diseño Lógico de Datos . . . . . 67
6.2.1. Introducción . . . . . 67
6.2.2. Metodología . . . . . 67

**7. Construcción del Sistema** **73**

**8. Pruebas del Sistema** **75**

**III Epílogo** **77**

**9. Manual de Implantación y Explotación** **79**

**10. Manual de Usuario** **81**

**11. Manual de Desarrollador** **83**

**12. Conclusiones** **85**
12.1. Bibliografía . . . . . 87

**13. Bibliografía** **89**

**14. Información sobre Licencia** **97**

# Índice de figuras


2.1. ***5e Character***: Pantalla de selección de clases . . . . . 15
2.2. ***5e Character***: Información de la clase *Druida* . . . . . 15
2.3. ***RPG Simple Dice***: Pantalla de selección de dados . . . . . 16
2.4. ***RPG Simple Dice***: Ejemplo de lanzamiento de dados . . . . . 16
2.5. ***BattleTrack***: Pantalla de grupos . . . . . 17
2.6. ***BattleTrack***: Ejemplo de combate . . . . . 17
2.7. ***RPGSound***: Menú principal . . . . . 17
2.8. ***RPGSound***: Menú de *Ambiente* . . . . . 17
2.9. ***RPG Character Sheet***: Pantalla de *Estado* . . . . . 18
2.10. ***RPG Character Sheet***: Pantalla de *Features* . . . . . 18
2.11. Patrón MVVM: *Modelo-Vista-Modelo de Vista*. Extraído de *Microsoft Docs* [14] . . . . . 20
2.12. Esquema de un modelo de datos. Extraído de *definición.de* [33] . . . . . 20
2.13. Modelo de desarrollo en cascada. Extraido de *(IONOS)*[38] . . . . . 21
2.14. Modelo de desarrollo en espiral. Extraído de *freepng.es* [39] . . . . . 22
2.15. Modelo de desarrollo con prototipos. Extraído de *sites.google.com* [41] . . 22
2.16. Modelo de desarrollo incremental. Extraído de *researchgate.net* [43] . . . . 23
2.17. Logo de *Scrum*. Extraído de *worldvectorlogo.com* [49] . . . . . 24
2.18. Ciclo de desarrollo de *Scrum*. Extraído de *Metodología Scrum* [50] . . . . 25
2.19. Roles en *Scrum*. Extraido de *herizont.com* [51] . . . . . 26

3.1. Planificación del proyecto . . . . . 32

4.1. Logo de *Android Studio* . . . . . 46
4.2. Logo de *Apache Jena* . . . . . 46
4.3. Logo de *React Native* . . . . . 46
4.4. Logo de *rdflib.js* . . . . . 46
4.5. Logo de *Xamarin* . . . . . 47

5.1. Modelo conceptual . . . . . 52
5.2. Diagrama de Casos de Uso del sistema . . . . . 53
5.3. Diagrama de Casos de Uso: Gestión de juegos . . . . . 53
5.4. Diagrama de Casos de Uso: Gestión de personajes . . . . . 57
5.5. Diagrama de Casos de Uso: Gestión de habilidades . . . . . 60
5.6. Modelo de Comportamiento:Pantalla de selección . . . . . 62
5.7. Modelo de Comportamiento: Crear personaje . . . . . 62
5.8. Modelo de Comportamiento: Editar personaje . . . . . 63
5.9. Modelo de Comportamiento: Eliminar personaje . . . . . 63
5.10. Modelo de Comportamiento: Cálculo de habilidades . . . . . 64

6.1. Estructura del patrón *MVVM*. Extraído de *medium.com* [69] . . . . . 66
6.2. Etapas de creación de *Ánima: Beyond Fantasy* . . . . . 70

# Índice de cuadros

1.1. *Lankoski, Heliö y Ekman*: Estructura ósea de una persona tridimensional. [5] . . . . . 6

3.1. Tabla de sueldos basados en roles . . . . . 35
3.2. Costes humanos del proyecto . . . . . 35
3.3. Costes de hardware . . . . . 36
3.4. Costes de software . . . . . 36
3.5. Costes varios . . . . . 37
3.6. Costes varios . . . . . 37
3.7. Coste total del proyecto . . . . . 37

4.1. Tabla de requisitos funcionales . . . . . 44
4.2. Comparativa entre *JavaScript* y *C#* . . . . . 47

5.1. Tabla de actores . . . . . 52

# Parte I

# Prolegómeno

*"En Arrswyd, la oscuridad ha consumido las estrellas en el cielo, dejando nada más que una sola ciudad en medio del olvido. Arrswyd es una nación amurallada donde permanece la última población del mundo. El día ha sido olvidado hace mucho tiempo, los miedos se hacen realidad, y entre las torcidas espirales, merodeando entre las sombras, hay horrores que se alimentan del miedo. El terror es todo lo que Arrswyd sabe ya que su gente es perseguida por criaturas y oprimida por el Señor inmortal, Yomon Ecuro. Algunos se animan a luchar contra el abismo y las maquinaciones del Ministerio y del Instituto, mientras que otros simplemente quieren vivir los últimos días del mundo hasta que se apague la última luz."*

---

*Arrswyd, Regnum Ex Nihilo*

# CAPÍTULO 1

# Introducción

## 1.1. Motivación

Los juegos de rol, a los que llamaremos ***RPG*** (***R**ole-**P**laying **G**ame*) a partir de ahora, son juegos con una meta particular, que es *"El objetivo de ambos tipos de juegos, computerizados y de otro tipo, es experimentar una serie de aventuras en un mundo imaginario, a través de un personaje avatar o un pequeño grupo de personajes cuyas habilidades y poderes crecen conforme el tiempo pasa."* [1].

Uno de los aspectos más importantes de los *RPG* son los personajes, ya que estos son la manifestación de los jugadores dentro del universo del juego, que a partir de ahora llamaremos **personajes jugadores** o ***PJs***. Ramos y Sueiro [2] plantean que, a diferencia del teatro, donde la elección de un personaje puede depender de las características físicas del intérprete, en los juegos de rol cualquier persona puede interpretar cualquier personaje, siempre que esté dentro de las posibilidades que ofrezca el juego en que se esté jugando. Los personajes deben definirse dentro de los limites establecidos por las reglas del juego, y en una historia como si se tratara de su biografía, explicando cómo ha sido la vida del personaje en la ambientación hasta el momento de comenzar la historia. A este proceso de definir un personaje para un jugador se conoce como **creación de personaje**.

Hay un amplio rango de posibilidades cuando tratamos la creación de *PJs* en diferentes mundos. En algunos juegos, el jugador sólo puede seleccionar algunas características predefinidas para el personaje, mientras que en otros juegos el usuario puede cambiar cada elemento del avatar [3]. Esto hace que la creación de personajes sea un proceso arduo y complejo, pues es necesario tener amplios conocimientos del juego para conocer todas las opciones de personalización disponibles para el jugador, y sus correspondientes características.

Egri [4] afirma en su libro *The Art of Dramatic Writing: Its Basis in the Creative Interpretation of Human Motives* que mientras todo objeto tiene tres dimensiones: profundidad, altura y anchura, los seres humanos tienen tres dimensiones adicionales: **fisiología**, **sociología** y **psicología**, y que sin el conocimiento de estas dimensiones, no se puede valorar a un ser humano. La primera dimensión, la **fisiología**, da color al punto de vista humano y le influye infinitamente, facilitando a una persona ser tolerante, desafiante, humilde o arrogante. La **sociología**, hace referencia a que no es posible realizar un análisis exacto de las diferencias entre una persona y el vecino de la puerta de al lado si no se tiene suficiente información de las circunstancias de ambas personas. La tercera dimensión, la **psicología**, es el producto de las otras dos, cuya influencia da vida a la ambición, la frustración, el temperamento, las actitudes y los complejos.

En base a esto, Lankoski, Heliö y Ekman [5] presentan una tabla, que se muestra a continuación, definiendo los aspectos de esta estructura definida por Egri, modificando y añadiendo algunos aspectos de la misma.

| ***Fisiología*** | ***Sociología*** | ***Psicología*** |
|---|---|---|
| Sexo | Clase | Estándares morales y vida sexual |
| Edad | Ocupación | Metas y ambiciones |
| Altura y anchura | Educación | Frustraciones y decepciones |
| Color de pelo, ojos y piel | Vida familiar | Temperamento |
| Postura | Religión | Actitud frente a la vida |
| Apariencia y rasgos distintivos | Raza y nacionalidad | Complejos y obsesiones |
| Defectos | Posición social | Imaginación, juicios, sabiduría, gustos y estabilidad |
| Rasgos hereditarios | Afiliaciones políticas | Extroversión, introversión y ambiversión |
| Físico | Entretenimientos y aficiones | Inteligencia |

Cuadro 1.1: *Lankoski, Heliö y Ekman*: Estructura ósea de una persona tridimensional. [5]

En lo referido a juegos de rol, *Tychsen*, *Hitchens* y *Brolund* sostienen que *"el diseño de personajes está dividido en cuatro componentes:* ***personalidad***, ***integración***, ***apariencia*** *y* ***reglas***, *que tienen su propio conjunto de restricciones para asegurar consitencia metodológica".* [6]

- Son las características basadas en las reglas del juego, tales como *atributos*, *aptitudes*, *habilidades* y *clase*. Los *RPG* proveen estas características en plantillas conocidas como *hojas de personaje*, que detallan todos los componentes basados en reglas del personaje, referentes al juego al que pertenezca el *PJ* en cuestión.

- ***Integración***: Los componentes de integración son aquellos que explican las circunstancias del personaje. Algunos de estos componentes son:

  - *Ubicación*: ¿Dónde está el personaje y por qué?
  - *Trasfondo*: ¿Cual es la historia detrás del personaje y qué eventos lo llevan al punto de inicio del juego?
  - *Contactos*: ¿Que contactos y relaciones tiene el personaje con otros personajes no controlados por jugadores (conocidos como *personajes no jugadores* o ***PNJs***)?
  - *Conexiones*: ¿Cuál es la relación entre el personaje y los demás *PJs*?

- ***Apariencia***: El personaje requiere una definición de su apariencia, para lo que son necesarios algunos rasgos de importancia:

  - *Representación*: ¿Cómo luce el personaje?
  - *Comportamiento físico*: ¿Cómo se comporta el personaje? ¿Cómo puede su interacción con el mundo modificar su personalidad o comportamiento?
  - *Interación*:¿Cómo interactua el personaje con el entorno y como interacciona con otros personajes?

- ***Personalidad***: Este componente incluye descripciones de la psique del personaje (emociones, comportamiento) y metas, con algunos rasgos de personalidad muy definidos enfáticamente (normalmente más sutiles y que dan lugar a la interpretación personal del jugador).

De todos los apartados previos, hay uno que destaca por estar plenamente orientado a la información del juego, que es el apartado de *Reglas*, ya que es el único que tiene información objetiva del personaje, y que sigue una estructura, dada por el *RPG* al que vaya a pertenecer el personaje, que posibilita establecer un conjunto de elementos estructurado, que unido a la normativa del juego, permita establecer las características técnicas de cualquier personaje.

En relación con lo comentado previamente, y de acuerdo con el autor del blog *Ars Rolica* [7], una de las novedades que se están popularizando más es la de los **generadores de personajes**, que son herramientas que permiten crear personajes de juegos sin necesidad de invertir grandes cantidades de tiempo, evitando posibles errores y deslices. Estos generadores pueden ser bastante útiles tanto para jugadores novatos que no conocen el sistema del juego, como para jugadores más experimentados que no puedan utilizar tiempo para documentarse completamente antes de crear su álter ego.

El objetivo de este proyecto consiste en diseñar e implementar un generador de personajes de juegos de rol de mesa, para lo que se perseguirán los siguientes subobjetivos:

- Realizar un estudio del arte de los *generadores de personaje* existentes, que permita conocer el estado actual de las herramientas que se encuentran en uso en la actualidad.
- Desarrollar un sistema que permita trabajar con diferentes *RPGs*, sin tener que salir de la aplicación, siempre y cuando se disponga de los ficheros que contengan la información de dichos juegos.
- Diseñar una estructura genérica para el sistema de información, de forma que permita a la aplicación poder adaptarse de la forma más completa posible a cada juego, permitiendo profundizar de la misma manera en juegos de alta complejidad que en juegos sencillos.
- Crear una interfaz de usuario intuitiva, que facilite al usuario la interacción con la aplicación y ésta resulte cómoda y agradable de utilizar.

## 1.2. Alcance

Este proyecto consiste en una aplicación que permita al usuario generar fichas técnicas de personajes para poder participar en partidas o campañas de *RPG*. La aplicación debe servir para poder desarrollar personajes de cualquier juego, siempre y cuando disponga de la información del mismo, pero a su vez debe poder explorar todas las posibilidades de cada juego, sin importar su complejidad.

El objetivo principal que se desea conseguir con esta aplicación es facilitar a todos los jugadores de rol una herramienta que facilite la comprensión del proceso de creación de personajes y reduzca la duración del mismo, pero sin dejar de valorar todas las opciones que el juego pone a su disposición.

El otro objetivo de este proyecto es desarrollar una herramienta general que pueda ser útil para cualquier *RPG*, independientemente del sistema de creación de personajes que utilice, pues no resulta práctico tener que disponer de un *generador de personajes* por cada *RPG* que existe, ya que el coste espacial de disponer de todos ellos resultaría imposible de contener en una herramienta de gama media, de forma que no estaría disponible para todos los jugadores.

Para conseguir esto, se ha considerado que, en vez de contemplar como objetivo la creación de una base de datos que contenga la información de un juego concreto, se va a fijar como meta el diseño de una estructura general para que cualquier persona que quiera introducir la información de un nuevo juego, tenga a su disposición una metodología clara para desarrollar el banco de datos de manera que la aplicación sea capaz de trabajar con éste.

## 1.3. Glosario de Términos

- ***Android***: *"Sistema operativo que se emplea en dispositivos móviles, por lo general con pantalla táctil. De este modo, es posible encontrar tabletas (tablets), teléfonos móviles (celulares) y relojes equipados con Android, aunque el software también se usa en automóviles, televisores y otras máquinas. Creado por Android Inc., una compañía adquirida por Google en 2005, Android se basa en Linux, un programa libre que, a su vez, está basado en Unix."* [8]

- ***C#***: *"Lenguaje de programación simple, moderno, orientado a objetos y fuertemente tipado. C# tiene sus raíces en la familia de lenguajes C [. . . ] C# está estandarizado por* ***ECMA*** *International con el estándar* ***ECMA-334*** *y por* ***ISO/IEC*** *con el estándar* ***ISO/IEC 23270****."* [9]

- ***Git***: *"Herramienta de control de versiones particularmente potente, flexible y bajos costes generales que hace del desarrollo colaborativo un placer. Git fue inventado por* ***Linus Torvalds*** *para dar soporte al desarrollo del kernel* ***Linux****, pero desde entonces ha probado ser valioso a un amplio rango de proyectos"* [10]

- ***GitHub***: *"Servidor de respositorios de código basado en el sistema de control de versiones* ***Git****"* [11]

- ***IDE***: *"Un entorno de desarrollo integrado o* ***IDE*** *es una aplicación visual que sirve para la construcción de aplicaciones a partir de componentes."* [12]

- ***Metodología de desarrollo***: *"Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en sus esfuerzos por implementar nuevos sistemas de información. Una metodología está formada por fases, cada una de las cuales se puede dividir en sub-fases, que guiarán a los desarrolladores de sistemas a elegir las técnicas más apropiadas en cada momento del proyecto y también a planificarlo, gestionarlo, controlarlo y evaluarlo."* [13]

- ***Metodología Ágil***: *"Metodologías que se derivan de la lista de los principios que se encuentran en el* ***Manifiesto Ágil****, y están basadas en un desarrollo iterativo que se centra que se centra más en capturar mejor los requisitos cambiantes y la gestión de los riesgos, rompiendo el proyecto en iteraciones de diferente longitud"* [13]

- ***Modelo***: *"Las clases de modelo son clases no visuales que encapsulan los datos de la aplicación. Por lo tanto, se puede considerar que el modelo representa el modelo de dominio de la aplicación, que normalmente incluye un modelo de datos junto con la lógica de validación y negocios."* [14]

- ***Modelo de Vista***: *"El modelo de vista implementa las propiedades y los comandos a los que la vista puede enlazarse y notifica a la vista de cualquier cambio de estado*

*a través de los eventos de notificación de cambios. Las propiedades y los comandos que proporciona el modelo de vista definen la funcionalidad que ofrece la interfaz de usuario, pero la vista determina cómo se mostrará esa funcionalidad."* [14]

- ***MVVM***: *"Patrón de arquitectura de software que ayuda a separar la lógica de negocios y presentación de una aplicación de su interfaz de usuario. "* [14]

- ***Ontología***: *"Las ontologías computacionales tienen como objetivo modelar la estructura de un sistema, por ejemplp, las entidades y relaciones relevantes que emergen de la observación, que son útiles a nuestros propósitos.[...] Gruber [15] describe la noción de una ontología como la especificación explícita de una conceptualización compartida"* [16]

- ***OWL***: *"El lenguaje de ontología web **OWL** es un lenguaje de marcado semántico para publicar y compartir ontologías en la World Wide Web. OWL está desarrollado como una extensión de vocabulario de **RDF** y está derivado del lenguaje de ontología web DAML+OIL."* [17]

- ***Protégé***: *"El sistema Protégé es un entorno para el desarrollo de sistemas basados en conocimiento que ha evolucionado durante más de una decada. Protégé comenzó comouna pequeña aplicación diseñada para un dominio médico (planificación de terapia basada en protocolos), pero ha evolucionado en un conjunto de herraminetas con un propósito mucho más general [...] El objetivo inicial de Protégé era reducir el cuello de botella de adquisición de conocimiento minimizando el rol del ingeniero de conocimiento construyendo bases de conocimiento."* [18]

- ***RDF***: *"El **F**ramework de **D**esripción de **R**ecursos o RDF es una fundación para procesar metadatos; provee interoperabilidad entre aplicaciones que intercambian información comprensible para máquinas en la Web. RDF enfatinza las facilidades para permitir el procesamiento automatizado de recursos web. "* [19]

- ***RDFS***: *"RDF y su esquema de extension, RDF Schema Specification (RDFS) forman las dos capas más bajas de la **Web Semántica**[. . . ] RDFS provee un mecanismo estándar para declarar clases y propiedades (globales) así como para definir relaciones entre clases y propiedades, usando sintaxis de RDF."* [20]

- ***RDFSharp***: *"RDFSharp es un framework ligero de **C#** que puede ser utilizado para realizar aplicaciones, servicios y sitios web capaces de modelar, almacenar y consultar datos **RDF/SPARQL**."* [21]

- ***Scrum***: *"Marco de trabajo para la gestión y desarrollo del software basada en un proceso iterativo e incremental utilizado comúnmente en entornos basados en el desarrollo ágil del software."* [22]

- ***SPARQL***: *"SPARQL es un lenguaje de consulta desarrollado pincipalmente para consultar grafos RDF."* [23]

- ***Sprint***: *"Período en el cual se lleva el desarrollo de una tarea."* [22]
- ***UML***: *"El Lenguaje Unificado de Modelado (UML) es, tal como su nombre lo indica, un lenguaje de modelado y no un método o un proceso. El UML está compuesto por una notación muy específica y por las reglas semánticas relacionadas para la construcción de sistemas de software. El UML en sí mismo no prescribe ni aconseja cómo usar esta notación en el proceso de desarrollo o como parte de una metodología de diseño orientada a objetos. "* [24]
- ***Vista***: *"La vista es responsable de definir la estructura, el diseño y la apariencia de lo que el usuario ve en la pantalla. Idealmente, cada vista se define en XAML, con un código subyacente limitado que no contiene la lógica de negocios. Sin embargo, en algunos casos, el código subyacente podría contener lógica de la interfaz de usuario que implementa el comportamiento visual que es difícil de expresar en XAML, como animaciones."* [14]
- ***Visual Studio***: *"Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para Windows, Linux y macOS [...] Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno compatible con la plataforma .NET (a partir de la versión .NET 2002). Así, se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos y videoconsolas, entre otros."* [25]
- ***Xamarin***: *"Xamarin es una plataforma de código abierto para compilar aplicaciones modernas y con mejor rendimiento para iOS, Android y Windows con .NET. Xamarin es una capa de abstracción que administra la comunicación de código compartido con el código de plataforma subyacente. Xamarin se ejecuta en un entorno administrado que proporciona ventajas como la asignación de memoria y la recolección de elementos no utilizados."* [26]
- ***Xamarin.Forms***: *"Xamarin.Forms es un marco de interfaz de usuario de código abierto. Xamarin.Forms permite a los desarrolladores compilar aplicaciones en Xamarin.Android, Xamarin.iOS y Windows desde un único código base compartido. Xamarin.Forms permite a los desarrolladores crear interfaces de usuario en XAML con código subyacente en C#. Estas interfaces se representan como controles nativos con mejor rendimiento en cada plataforma."* [27]
- ***XML***: *"XML is a markup language for documents containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents."* [28]

- ***XAML***: *"XAML significa **L**enguaje de **M**arcado de **A**plicaciones **E**xtensible. Es el nuevo lenguaje declarativo de Microsoft para definir aplicaciones con interfaces de usuario. XAML provee una sintexis accesible, extensible y localizable para definir interfaces de usuario separadas de la lógica de la aplicación. smiliar a la técnica orientada a objeto para desarrollar aplicaciones de múltiples niveles con la arquitectura Modelo-Vista-Controlador."* [29]

- ***Web Semántica***: *"Extensión de la actual web en la que a la información disponible se le otorga un significado bien definido que permita a los ordenadores y las personas trabajar en cooperación. Se basa en la idea de tener datos en la web definidos y vinculados de modo que puedan usarse para un descubrimiento, automatización y reutilización entre varias aplicaciones."* [30]

## 1.4. Estructuración del documento

El documento que se presenta está estructurado en una colección de capítulos, en los que se describen de manera precisa y detallada todas y cada una de las etapas por las que ha pasado el proyecto, desde su inicio hasta su conclusión. A continuación se muestra un breve resumen de los contenidos de cada capítulo:

- ***Capítulo 1. Introducción***. El capítulo inicial consiste en una introducción al proyecto, explicando los antecedentes a su desarrollo, así como la motivación para ponerlo en práctica, los objetivos que debe cumplir y un glosario de términos para facilitar la comprensión del presente documento.

- ***Capítulo 2. Antecedentes***. Tras la introducción, se abordan los fundamentos necesarios para simplificar la lectura de este documento. A su vez, se exponen las diferentes tecnologías de las que se han aplicado para la consecución y puesta en marcha de este proyecto.

- ***Capítulo 3. Planificación del proyecto***. Este capítulo engloba toda la información relevante a aspectos de suma importancia tales como la evaluación de riesgos o la planificación temporal del proyecto.

- ***Capítulo 4-8. Análisis, Diseño, Codificación y Pruebas***. Este conjunto de capítulos profundizan en los diversos aspectos y fases que comprenden el desarrollo de un producto haciendo uso de una *metodología ágil*. Esto posibilita analizar y enriquecer el producto en su desarrollo, mejorando así el resultado final.

- ***Capítulo 9. Manual de Instalación***. Aquí se ha desarrollado un documento con las instrucciones necesarias para realizar la instalación de la aplicación.

- ***Capítulo 10. Manual de Usuario***. Se ha redactado un manual de usuario para la aplicación, cuyo objetivo es garantizar un uso eficiente y responsable de la misma por parte de los usuarios.

- ***Capítulo 11. Conclusiones***. El último capítulo es un breve repaso sobre el desarrollo del proyecto, que incluye una opinión personal y un apartado de posibles mejoras que se podrían realizar al proyecto en un futuro.

# CAPÍTULO 2

# Antecedentes

## 2.1. Estado del Arte

Con el objetivo de simplificar el proceso de creación de personajes en los juegos de rol tradicionales, se han originado multitud de aplicaciones con diferentes funciones y finalidades.

Algunas de las aplicaciones son referencias completas de los juegos, que sirven como elementos de consulta accesibles, rápidos y precisos. Un ejemplo de ello es ***5e Character***, que es una referencia completa de personajes para *Dragones y Mazmorras, 5ª Edición.*


Figura 2.1: ***5e Character***: Pantalla de selección de clases

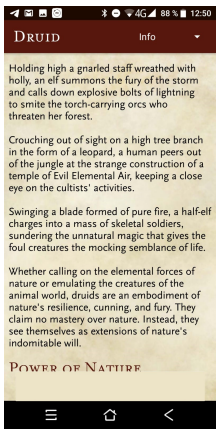


Figura 2.2: ***5e Character***: Información de la clase *Druida*

Estas aplicaciones ofrecen información del juego de manera accesible, pero no permiten aprovechar la digitalización de ésta para otros usos. Permiten realizar búsquedas rápidas, pero tampoco suelen disponer de toda la información del juego, de manera que para poder hacer consultas del juego completo es necesario acceder al libro del juego o hacer uso de más aplicaciones. Además, son aplicaciones que sólo pueden ser útiles para una versión específica del juego, ya que las versiones nuevas de los juegos de rol suelen incluir una gran cantidad de cambios. Aún así, éstas pueden no resultar obsoletas al momento de surgir la nueva versión, pues suele haber un período de transición entre versiones bastante amplio.

Como en los juegos de rol se utilizan diferentes tipos de dados (de 3, 4, 6, 8, 10, 12, 20, 30 y hasta 100 lados), también podemos encontrar aplicaciones como ***RPG Simple Dice***, que están preparadas para simular los resultados de lanzar estos dados, lo cual hace que se conviertan en una herramienta útil si no disponemos de alguno de éstos.

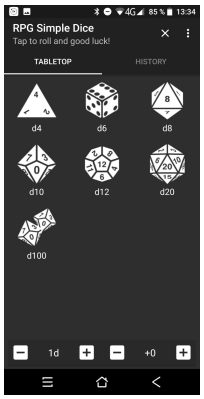


Figura 2.3: ***RPG Simple Dice***: Pantalla de selección de dados

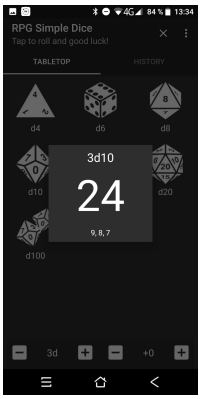


Figura 2.4: ***RPG Simple Dice***: Ejemplo de lanzamiento de dados

Otras aplicaciones proporcionan algunas herramientas que simplifican cálculos que resultan tediosos durante el transcurso de la partida, como es el caso de ***BattleTrack***. Dependiendo de la aplicación que se utilice, puede incluir su propio simulador de dados, lo que hace innecesario el uso de aplicaciones específicas para ello.

El punto negativo de estas aplicaciones es que son generales y no disponen de la información específica de los personajes de los jugadores, de manera que el resultado es aproximado, o requiere que el usuario especifique previamente toda la información, lo que puede llegar a resultar tedioso si en una partida se realizan múltiples cálculos de éste tipo.

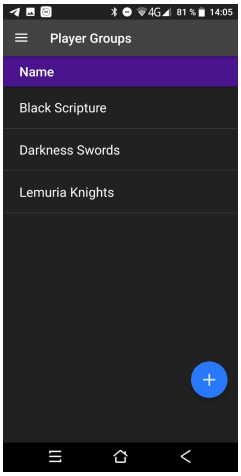


Figura 2.5: ***BattleTrack***: Pantalla de grupos

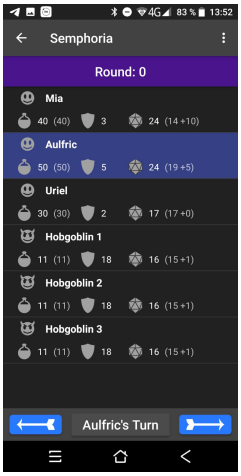


Figura 2.6: ***BattleTrack***: Ejemplo de combate

Con el fin de ayudar en la ambientación, aplicaciones como ***RPGSound*** aportan bibliotecas de sonido que se pueden utilizar durante la representación de la partida para sumirse en ella.

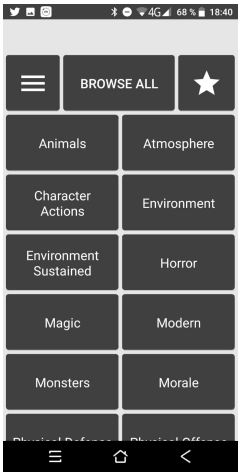


Figura 2.7: ***RPGSound***: Menú principal


Figura 2.8: ***RPGSound***: Menú de *Ambiente*

Finalmente, quedan las aplicaciones conocidas como *generadores de personaje*, que permiten al usuario crear personajes para formar parte de una partida de rol, y acceder a esa información de forma rápida. Un buen ejemplo de esto es ***RPG Character Sheet***. Aunque estas aplicaciones suelen reunir las funcionalidades de la mayoría de las aplicaciones previamente explicadas, suelen tener una interfaz bastante caótica, lo que resulta en una curva de aprendizaje bastante elevada para usuarios nuevos, que acaban prefiriendo el método tradicional, o el uso de varias aplicaciones más sencillas.

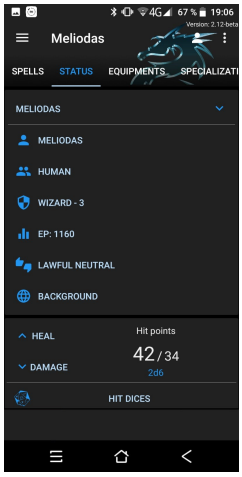


Figura 2.9: ***RPG Character Sheet***: Pantalla de *Estado*

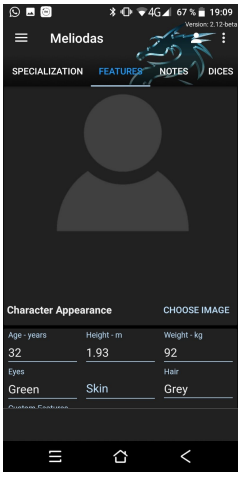


Figura 2.10: ***RPG Character Sheet***: Pantalla de *Features*

En resumen, hay un amplio abanico de aplicaciones que buscan resultar de utilidad a los jugadores de *RPG* de diferentes maneras, ya sea facilitando la búsqueda de información, realizando los cálculos de los valores finales de las tiradas de dados o generando la información de los personajes que forman parte de la partida.

## 2.2. Crítica al estado del arte

Tal y como se ha comentado previamente, existe un holgado abanico de aplicaciones cuya meta es mejorar y/o simplificar aspectos en lo referente a los juegos de rol, y aunque cumplen con su propósito, a veces no resultan tan efectivas como debieran.

Esto puede deberse a que tras dedicar el tiempo y esfuerzo necesarios para desarrollar la aplicación, el estudio del juego ha aprovechado ese tiempo de producción para revisar el juego y editarlo, realizando modificaciones que provocan que ***la aplicación quede***

***obsoleta en poco tiempo***. Como las aplicaciones suelen estar orientadas a una versión específica de un juego en concreto, éstas dejan de ser útiles cuando se desea utilizar una versión diferente del juego, o un juego diferente, lo que resulta en un mercado muy amplio de aplicaciones muy específicas, mientras que es muy difícil, casi imposible incluso, encontrar una que dé soporte a varios juegos a la vez, o a diferentes versiones del mismo juego.

Otro inconveniente es que las aplicaciones que requieren mucha información específica, como los generadores de personaje, pueden llegar a ***resultar muy complejas***, y al tener una interfaz poco intuitiva, provoca que el usuario no experimentado considere que el esfuerzo que tiene que dedicar para aprender cómo utilizarla es mayor que el de realizar el proceso manualmente.

También existen aplicaciones que no contemplan la reutilización de la información que han producido para efectuar operaciones que mejoren la jugabilidad, por lo que el usuario no le ve provecho a emplear dichas aplicaciones.

## 2.3. Propuesta

Tras analizar el estado del arte, y destacar algunos de los aspectos negativos de éste, lo que prosigue es diseñar una propuesta que permita recoger las mejores ideas de las herramientas actuales, y que además pueda dar solución a los problemas comentados en el apartado 2.2 (*Crítica al estado del arte*).

### 2.3.1. Plataforma

En primer lugar, se debe considerar cuál será la plataforma en la que se implementará la aplicación, puesto que es la base sobre la que se realiza el desarrollo, y que condiciona las herramientas que se pueden utilizar para la elaboración del proyecto.

Como se puede apreciar en la sección 2.1 (*Estado del arte*), muchas de las aplicaciones existentes se desarrollan en entornos móviles, lo que supone que la aplicación pueda ser accesible desde cualquier sitio. Por otro lado, se considera una opción plausible el poder extender la aplicación a otros entornos, tales como sistemas de sobremesa, o el entorno web. Teniendo todo esto en consideración, se ha tomado la decisión de realizar una **aplicación móvil**, haciendo uso de **herramientas de desarrollo multiplataforma**, de manera que en un futuro sea posible extender el proyecto a otras plataformas.

### 2.3.2. Arquitectura

Después de seleccionar la plataforma principal de desarrollo, el siguiente paso es definir la arquitectura de la aplicación, que hace referencia a *"la estructura general de este y a las formas en las que ésta da integridad conceptual a un sistema."* [31]. Como explica Cardacci [32], con el tiempo se han ideado una amplia variedad de formas arquitectónicas, con el objetivo de separar los diferentes desafíos que plantea el desarrollo

de software conceptualmente, y lograr cierto grado de independencia entre los elementos que componen el software, de manera que los cambios realizados en uno no tenga efecto en el resto.

De entre todas las posibilidades existentes, hemos optado por utilizar el patrón *Modelo-Vista-Modelo de Vista* (**MVVM**) [14], que separa el software en tres componentes: *modelo*, *modelo de vista* y *vista*.

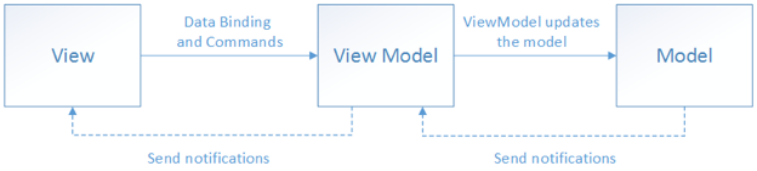


Figura 2.11: Patrón MVVM: *Modelo-Vista-Modelo de Vista*. Extraído de *Microsoft Docs* [14]

### 2.3.3. Modelo de datos

Como definen Pérez y Gardey [33], un modelo de datos es un esqueleto estructural que permite organizar y documentar la información. Esta estructura busca definir el acceso y el almacenamiento de los datos de forma unívoca, de manera que permitan la comunicación entre las aplicaciones que hagan uso de los datos del modelo.

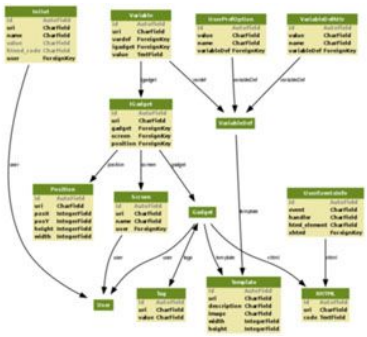

Figura 2.12: Esquema de un modelo de datos. Extraído de *definición.de* [33]

Para este proyecto hace falta un modelo de datos que permita representar diversos mundos, y establecer relación entre sus elementos. Por ello, se ha tomado la decisión de plantear un modelo **ontológico**. Citando a Muñoz y Aguilar [34], *"una ontología es una base de datos que describe los conceptos del mundo en un dominio específico, algunas de sus propiedades y cómo estos conceptos se relacionan entre sí"*. [34]

## 2.4. Metodología

### 2.4.1. Metodologías de desarrollo software

El desarrollo de software está en constante cambio. Esto se debe en parte a la continua aparición de nuevas tecnologías que transforman los modelos teóricos vigentes. Por otro lado, existe una barrera entre las herramientas de desarrollo y la metodología que impide la puesta en práctica de muchos de los modelos propuestos. No es fácil adaptarse de manera adecuada a una metodología de desarrollo de software, lo que resulta en un proceso con posibles demoras. No obstante, *el uso de una metodología adecuada ha probado ser un pilar para el desarrolllo de un proyecto de construcción de software* [35]

De aquí es posible extraer dos ideas claras: la primera es que adaptarse a una metodología es una tarea complicada, pero que de lograrse con éxito, son claros los beneficios obtenidos frente a los resultados si no se hubiera realizado dicha adaptación. La segunda es que resulta necesario realizar un estudio para conocer cuáles son las métodologías existentes, cuáles están presentes en el mercado, conocer sus ventajas e inconvenientes, conocer su proceso de implementación y conocer si su alcance está alineado con el objetivo que se desea lograr.

Actualmente existe un gran abanico de metodologías, las cuales se adaptan en mayor o menor medida al tipo de producto que se pretende desarrollar. La gran mayoría de ellas están basadas en alguno de los siguientes modelos de desarrollo de software:

- **Desarrollo en cascada**: Pressman [36] plantea el desarrollo en cascada como el método que ordena las etapas de desarrollo de forma directa, de manera que cada etapa se inicia tras la finalización de la anterior.

  Como expresa Prieto [37], aplicar esta metodología muestra beneficios cuando cada fase de desarrollo dispone del tiempo necesario para llevarse a cabo. Prieto también comenta que para reducir los riesgos, es necesario tener establecidos los requisitos del proyecto claramente desde el principio del proyecto.

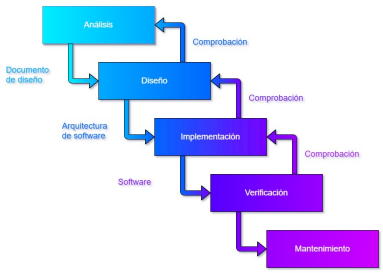


Figura 2.13: Modelo de desarrollo en cascada. Extraido de *(IONOS)*[38]

- **Desarrollo en espiral**: Este modelo permite un análisis más profundo de las etapas del desarrollo del producto, de acuerdo con Prieto [37]. Las actividades del modelo están dispuestas en espiral, en la que cada iteración o vuelta representa

un conjunto de actividades acorde a una etapa del desarrollo. El orden de las actividades queda determinado por el análisis del riesgo, comenzando por el bucle interior,

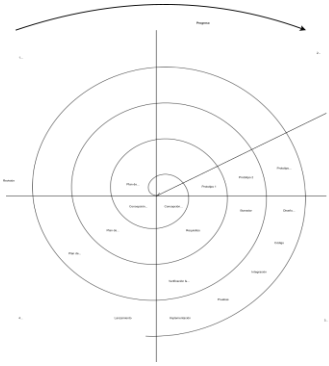

Figura 2.14: Modelo de desarrollo en espiral. Extraído de *freepng.es* [39]

- **Desarrollo con prototipos**: Desde el punto de vista de Roque y col. [40], el desarrollo con prototipos mejora la comunicación entre el equipo de desarrollo y el cliente. Esto permite que se puedan introducir cambios con facilidad y disminuya el tiempo de desarrollo. Consiste en un proceso iterativo que tiene cinco fases:

  1. ***Comunicación***: Se indica un conjunto de objetivos que el software debe cumplir.
  2. ***Plan rápido***: Se propone una estrategia para llevar a cabo el desarrollo
  3. ***Diseño rápido***: Se realiza el diseño de una interfaz gráfica rápidamente.
  4. ***Construcción***: Se construye el prototipo del sistema software.
  5. ***Entrega y retroalimentación***: Se entrega el prototipo y el cliente realiza una retroalimentación al equipo, que da inicio a una nueva iteración que incorpora los ajustes indicados en la información dada por el cliente.

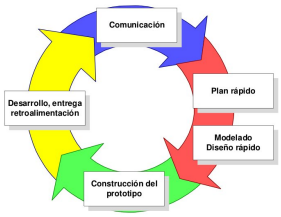


Figura 2.15: Modelo de desarrollo con prototipos. Extraído de *sites.google.com* [41]

- **Desarrollo incremental**: Zumba [42] postula que el desarrollo incremental está basado en la introducción por etapas (o iteraciones) de funcionalidades de la aplicación. La primera iteración suele ser básica, con los elementos mínimos requeridos, y cada iteración posterior es una versión entregable del proyecto, que introduce alguna funcionalidad nueva.

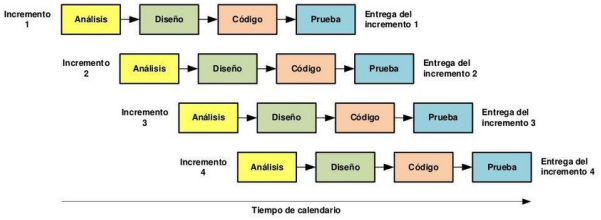


Figura 2.16: Modelo de desarrollo incremental. Extraído de *researchgate.net* [43]

### 2.4.2. Metodologías Ágiles

*"En febrero de 2001 nace el término* ***ágil*** *aplicado al desarrollo de software, tras una reunión celebrada en* Utah (EEUU). *El objetivo de la misma fue esbozar valores y principios que deberían permitir desarrollar software de manera rápida, dando respuesta a los cambios surgidos durante el desarrollo del proyecto. Se pretende con esto ofrecer alternativas a los procesos de desarrollo software tradicionales, rígidos y dirigidos por la documentación que se generaba en cada una de las etapas del proceso."* [13]

El punto de partida para ello fue el **Manifiesto Ágil** [44]: documento que resume la filosofía ágil, en el cual se valoran los siguientes elementos:

- **El individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas**: Las personas que forman parte del proyecto son el principal factor de éxito de un proyecto, de manera que el entorno influye menos que la bondad del equipo que realiza el desarrollo. Es preferible que el entorno se adapte al equipo.

- **Desarrollo de software útil es mejor que conseguir una buena documentación**: La regla a seguir es *producir sólo documentos que sean necesarios inmediatamente para tomar una decisión importante.*

- **La colaboración del cliente sobre la negociación de un contrato**: Se propone una interacción constante entre cliente y desarrolladores, de manera que esta colaboración permita marcar el ritmo del proyecto y asegure el éxito del mismo.

- **Respuesta rápida a los cambios es mejor que seguir un plan de forma estricta**: La habilidad de responder a los cambios determina el éxito o fracaso del proyecto, de manera que lo más importante de la planificación es su flexibilidad.

Los principios del *Manifiesto Ágil* se basan en estos valores, que a su vez hacen de fundamentos de todas las metodologías ágiles, orientando el desarrollo a la rápida obtención de un producto funcional aunque no tenga todas sus funciones implementadas.

Del modelo de desarrollo ágil se pueden encontrar diversas metodologías, como **eXtreme Programming** [45], **Scrum** [46] o **Crystal Clear** [47]. Para este proyecto se ha tomado la decisión de seguir la metodología ***Scrum*** ya que debido a la naturaleza y complejidad del mismo, es posible que sea necesario realizar cambios en el planteamiento del proyecto durante el proceso de desarrollo.

### 2.4.3. SCRUM

#### Introducción

Empleando las palabras de Rodríguez, *"**SCRUM** es una de las metodologías de desarrollo ágil más reconocidas a nivel mundial, su concepción resulta de unos análisis realizados por **Ikujiro Nonaka** e **Hirotaka Takeuchi** en los años 80, resaltando el trabajo en equipo para el desarrollo de productos y la autonomía que estos deben tener (Takeuchi & Nonaka, 1986). Su diseño se debe a que en los años 90, **Jeff Sutherland** y **Ken Schwaber** formalizaron un marco de trabajo y unas reglas aplicadas particularmente al desarrollo de software de productos complejos."* [48]


Figura 2.17: Logo de *Scrum*. Extraído de *worldvectorlogo.com* [49]

#### Características

A continuación se muestra una serie de características que deben tener todos los procesos que se introducen al marco de la metodología *Scrum*:

- El desarrollo incremental de los requisitos en bloques temporales cortos y fijos.
- Se da prioridad los requisitos más valorados por el cliente.
- El equipo se sincroniza diariamente y se realizan las adaptaciones necesarias.
- Tras cada iteración se muestra el resultado real al cliente, para que tome las decisiones necesarias en relación al resultado observado.

- Se le da al equipo la autoridad necesaria para poder cumplir los requisitos.
- Fijar tiempos máximos para lograr objetivos.
- Equipos de trabajo pequeños (de 5 a 9 personas).

### Ciclo de desarrollo

Para entender el ciclo de desarrollo de *Scrum* es necesario conocer las fases que lo definen:

1. **Planificación**: Reunión de los involucrados en la que se definen los requisitos prioritarios para la iteración actual y se elabora una lista de tareas necesarias para lograr los requisitos previamente seleccionados.

2. **Scrum diario**: Evento del equipo de desarrollo de quince minutos, que se realiza diariamente durante la ejecución de la iteración para explicar lo que se ha alcanzado desde la última reunión, lo que se hará antes de la siguiente y los obstáculos que se han presentado.

3. **Revisión**: El equipo presenta al cliente los requisitos completados en la iteración. En función de los resultados mostrados y de los cambios habidos en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, replanificando el proyecto.

4. **Retrospectiva**: El equipo analiza cómo ha sido su manera de trabajar y qué problemas podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad.

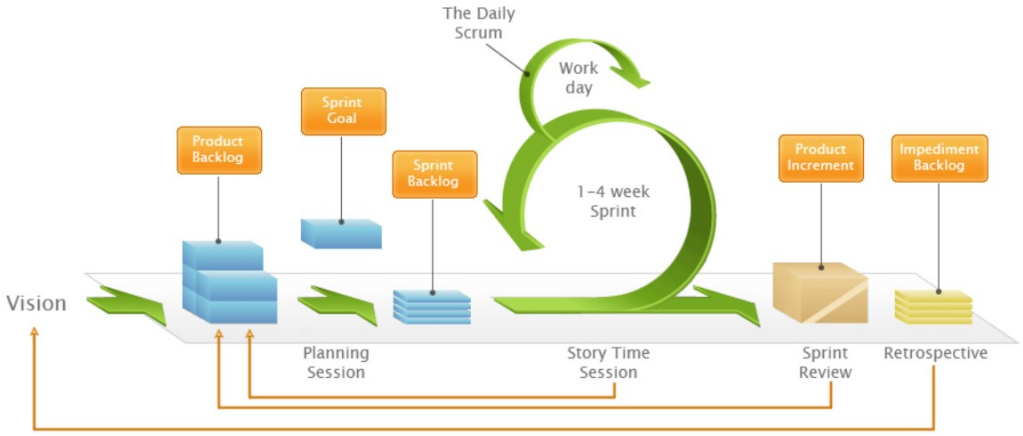


Figura 2.18: Ciclo de desarrollo de *Scrum*. Extraído de *Metodología Scrum* [50]

### Roles

Los roles presentes en *Scrum* son los siguientes:

- **Product Owner**: Tiene la responsabilidad de decidir qué trabajo necesita hacerse, y maximizar el valor del proyecto o producto que se esté llevando a cabo. Para ello debe tener las siguientes cualidades:

  1. *Saber gestionar prioridades*: Es responsable de gestionar los presupuestos, de contratar al equipo de desarrollo y de explicar cuál es el valor que produce el producto en el que está invirtiendo.
  2. *Toma de decisiones*: Debe ser capaz de tomar decisiones por su cuenta.
  3. *Coordinador*: Tiene que poder medir el valor generado y utilizar la flexibilidad de entregar cada *sprint* para incrementar ese valor.

- **Scrum Master**: Persona que ayuda al equipo y a la organización a optimizar el uso de la metodología. Traslada la visión del proyecto al equipo, y elimina los obstáculos que impiden que el equipo alcance el objetivo del *sprint*.

- **Development Team**: Grupo de profesionales con los conocimientos técnicos necesarios y que desarrollan el proyecto de manera conjunta llevando a cabo los requisitos a los que se comprometen al inicio de cada *sprint*.

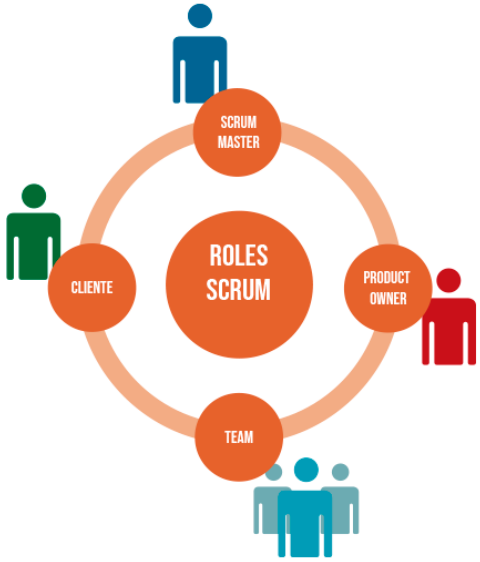


Figura 2.19: Roles en *Scrum*. Extraido de *herizont.com* [51]

## 2.5. Resumen

Las aplicaciones actuales permiten generar información útil para los jugadores de rol, tal como referencias rápidas, tiradas de dados, resultados de combates e incluso la información de sus personajes. Pero la mayoría de ellas están limitadas en que su ámbito suele estar reducido a una única versión de un juego específico. Además, no pueden personalizarse para el jugador, o requieren que se introduzca la información a mano cada vez que se utilizan, resultando ser menos práctico que el método tradicional.

En relación a esto, es de gran interés promover el uso de ontologías en este tipo de aplicaciones, como este proyecto, por ejemplo, pues permiten relacionar elementos de diferentes ontologías entre sí, permitiendo reutilizar elementos ya definidos, y ampliar modelos de datos sin realizar modificaciones de las versiones previas, posibilitando el uso de nuevas versiones de los modelos sin alterar los ya existentes, facilitando al usuario la posibilidad de escoger la versión que considere más adecuada para cada ocasión.

Por otro lado, con respecto al desarrollo de este proyecto se ha decidido optar por la metodología *Scrum* porque permite al equipo de desarrollo trabajar con requisitos concretos a corto plazo, priorizando los requisitos más valorados por los clientes, y que permite a éstos últimos disponer de resultados al final de cada iteración, de manera que pueden tomar las decisiones que consideren oportunas para la siguiente iteración del desarrollo.

# CAPÍTULO 3

# Planificación del proyecto

## 3.1. Planificación del proyecto

En este apartado se van a establecer el orden de las tareas que se deben realizar y la estimación del tiempo que se considera necesario para la elaboración de este proyecto. Finalmente se mostrará un análisis de la desviación temporal que indicará la diferencia entre los tiempos estimados con los empleados realmente.

### 3.1.1. Objetivo inicial

El objetivo inicial de este proyecto era desarrollar una aplicación móvil para la generación y manipulación de información de personajes de juegos de rol, extensible a cualquier juego de rol cuya información esté almacenada en la aplicación.

### 3.1.2. División del trabajo

La estructura de desglose del trabajo está dispuesta a continuación:

- **Planificación**: Inicialmente, se realiza una planificación *grosso modo*, que facilita una visión global de los hitos que se deben conseguir, y los pasos adecuados para alcanzarlos.
- **Conocimientos**: Al inicio del proyecto, el equipo de desarrollo de este proyecto no estaba relacionado con algunas de las herramientas que han sido utilizadas. Por otra parte, otras herramientas sí eran conocidas por el equipo, pero no con la profundidad que ha exigido el proyecto. Es precisamente esto lo que ha motivado al equipo a recuperar la información que ya obtenía sobre las herramientas y ampliarla mediante la búsqueda y lectura de documentación que pudiera beneficiar en el desarrollo.

- **Desarrollo**: Tras un breve período para ponerse al día con los conocimientos básicos para comenzar el proceso de desarrollo, se realizarán los diferentes *sprints* con las siguientes tareas:

  - ***Planificación***: Cada *sprint* tendrá su planificación específica seleccionando los requisitos *necesarios* para su desarrollo con el cliente. Como este proyecto *no tiene un cliente claramente definido, se ha entrevistado a* ***potenciales clientes*** de la aplicación para poder conocer las necesidades que consideran oportunas como posibles usuarios de la aplicación. Posteriormente, se procede a la planificación del *sprint* elaborando una lista de tareas que deben llevarse a cabo para considerar que se ha finalizado el *sprint* de manera satisfactoria.
  - ***Análisis***
  - ***Diseño***
  - ***Implementación***: Se realizarán pruebas de funcionalidad de manera simultánea a la codificación del proyecto, pudiendo comprobar que las funciones finalizadas cumplen su función de forma correcta y completa, sin errores.
  - ***Documentación***: Al finalizar cada *sprint* se actualizará la documentación del proyecto.

- **Memoria**: Aunque la memoria se ha realizado de forma conjunta con el proyecto desde su inicio, algunos aspectos como los manuales, han requerido que otros aspectos del proyecto estuvieran finalizados para poder realizarse.

- **Presentación**: Elaboración de una presentación para la defensa del proyecto ante el tribunal.

### 3.1.3. Identificación de *sprints* y estimación de tiempos

El equipo de desarrollo procede, en este punto de la memoria, a describir las iteraciones que se han dado a lo largo de la elaboración del proyecto.

1. ***Sprint 0:*** **Planificación inicial**: Forman parte de esta iteración tanto las reuniones con los clientes potenciales, como el estudio y elección de las tecnologías a emplear.

2. ***Sprint 1:*** **Preparación del equipo**: Durante el primer *sprint* se pondrá a punto el entorno de trabajo, instalando las aplicaciones necesarias para el desarrollo. También se procederá a la instalación de aplicaciones que, no siendo necesarias, serán de ayuda durante el proceso.

3. ***Sprint 2:*** **Creación de la ontología**: Este *sprint* es el más extenso de todos, debido a que abarca desde el esbozo inicial del modelo conceptual de la ontología, que debe contener toda la información de un juego de rol, hasta el momento en que ésta queda totalmente operativa. Como el sistema debe poder procesar otras

ontologías además de la elaborada por el equipo que lleva a cabo este proyecto, no se ha considerado finalizado el sprint hasta que se han realizado todas las modificaciones necesarias, tanto en la lógica de negocio de la aplicación como en la ontología, para que sea posible procesar y disponer correctamente de toda la información de la ontología.

4. ***Sprint 3:*** **Creación de personajes**: El objetivo principal de la aplicación se abordará en este *sprint*, ya que requiere que la lógica de negocio de la aplicación acceda a la ontología de un juego de rol, y extraiga información de esta para mostrar un formulario de creación de personajes paso a paso, personalizado para el juego de rol en cuestión, y que se genere un fichero con la información del personaje que ha sido seleccionada por el usuario en el formulario.

5. ***Sprint 4:*** **Visualización, modificación y eliminación de personaje**: La aplicación tiene que poder mostrar la información del personaje creado previamente, y permitir al usuario editar parte de la misma si así lo desea.

6. ***Sprint 5:*** **Validador de ontologías**: El proyecto dispondrá de una aplicación de consola que permita comprobar a un desarrollador si la ontología del juego que está elaborando podría funcionar en la aplicación principal del proyecto.

7. ***Sprint 6:*** **Cálculo de habilidades de personaje**: La aplicación podrá realizar cálculos con la información almacenada en la hoja de personaje. Este campo se considera opcional, debido a que al tener una fecha límite y un proceso complejo, es posible que no se pueda alcanzar este objetivo, el cual quedaría pendiente como trabajo futuro.

8. ***Sprint 7:*** **Pruebas**: Se realizarán pruebas para comprobar que la aplicación cumple todos los objetivos del proyecto.

9. ***Sprint 8:*** **Documentación**: La aplicación dispondrá de varios manuales que permitan a los usuarios conocer sus funciones y la manera correcta de utilizarlas.

### 3.1.4. Resumen de la planificación del proyecto

A continuación se muestra un resumen de la planificación previamente descrita desglosada según las actividades a realizar en un diagrama de Gantt.

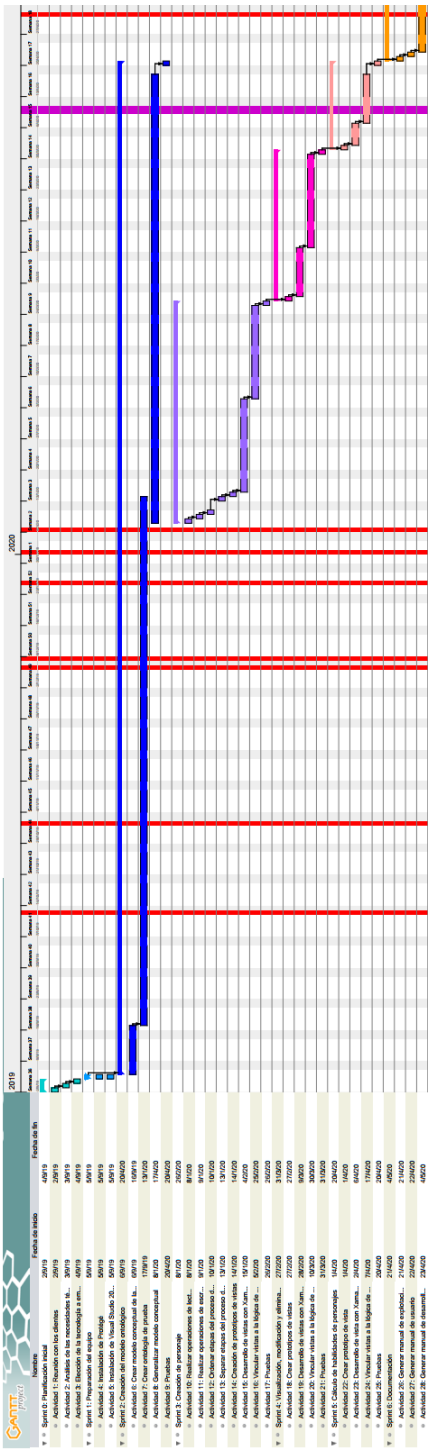

Figura 3.1: Planificación del proyecto

Los tiempos estimados han estado expuestos a imprevistos, ya fueran de tipo técnicos o tecnológicos:

- *Librería* **dotNetRDF** *no permite trabajar con propiedades del formato* **OWL**: La primera librería que el equipo encontró para poder trabajar con el formato *RDF* fue **dotNetRDF**. El problema sucedió cuando, una vez con el proyecto encaminado, resulta que la librería no es compatibles con las propiedades del formato *OWL*, de manera que no era posible realizar tuplas *sujeto-predicado-objeto*, y por tanto, no es una librería compatible con la aplicación que el equipo tiene como objetivo elaborar. Esto supuso un retraso de al menos un mes de trabajo para el equipo, ya que no sólo se había perdido el tiempo dedicado en aprender a utilizar la librería, sino que era necesario buscar otra que permitiera realizar aquello que la primera no podía, lo que podría hacer retroceder todo el proyecto, ya que de no encontrar una, habría que cambiar las herramientas base para el desarrollo. Después de buscar bastante, el equipo encontró la librería **RDFSharp**, que aún estando en desarrollo, sí es compatible con las propiedades necesarias para realizar tuplas *sujeto-predicado-objeto*, y por tanto, permite trabajar con el formato *OWL*, siendo así compatible con el producto final.

- *Error en el SO causa error en repositorios del proyecto*: Durante una de las actualizaciones del proyecto, el sistema operativo del ordenador que realizaba la actualización, causó la corrupción de gran parte de los ficheros del proyecto, resultando afectados los repositorios local y remoto del proyecto. El equipo entonces hizo uso de una copia de seguridad almacenada en un dispositivo externo, para recuperar la mayor cantidad de trabajo posible, pero al no estar actualizado completamente, fue necesario invertir entre 7 y 8 horas para recuperar el trabajo perdido.

## 3.2. Organización

### 3.2.1. Agentes involucrados

Las personas involucradas en la elaboración de este proyecto son:

- Los directores (o tutores) del proyecto: La labor de los directores del proyecto consiste en revisar el estado del proyecto durante todo el proceso de desarrollo y contribuir con las ideas que estimen oportunas para perfeccionarlo.

- Los clientes, que son los potenciales usuarios finales de la aplicación. Ellos indican los requerimientos que consideran necesarios para que el producto final sea atractivo para el público objetivo (*target*) del producto.

- El equipo de desarrollo, formado en este caso por una persona, que se encarga de la elaboración del proyecto.

### 3.2.2. Roles

En esta sección se van a abordar los diversos roles que forman parte del equipo de desarrollo de este proyecto:

- **Business Analyst**: Es la persona que se encarga de definir las funcionalidades y características del producto, priorización y refinamiento del *backlog* (lista de trabajos pendientes). [52]
- **Solution Architect**: Es el encargado de garantizar la integridad técnica y la coherencia de la solución del proyecto. En este caso se considera contar con uno pues se utiliza tecnología cuya implementación no ha sido probada con la funcionalidad deseada para el proyecto actual. [53]
- **UX Designer**: Es el responsable de elaborar el diseño de cada vista con la que interactúa el usuario. [54]
- **Software Developer**: Es la persona encargada de convertir la especificación del sistema a código funcional. [55]
- **Tester**: Es el encargado del apartado de pruebas del proyecto, desde la identificación de las condiciones de las pruebas, hasta su automatización, pasando por su creación y la especificación de los procesos de prueba. [56]
- **Documentalist**: Es la persona especializada en ayudar a investigadores y desarrolladores en su búsqueda de documentación científica y/o técnica. [57]

### 3.2.3. Recursos utilizados

Los recursos que han sido utilizados en el desarrollo de la aplicación son:

- Un ordenador personal, en el que se ha constituido el entorno de trabajo.
- Visual Studio 2019 Community, como entorno de desarrollo para la aplicación.
- Protégé, un editor de ontologías de código abierto
- Xamarin, como plataforma para el desarrollo de aplicaciones multiplataforma.
- LaTeX, como herramienta para la elaboración de la memoria.
- Dispositivo móvil con *Android*, para probar que la aplicación funciona en la plataforma deseada.

Como todos los recursos software previamente citados son gratuitos, su uso no supone coste alguno en licencias de aplicaciones. No se tendrá en cuenta el precio del ordenador personal en los costes del proyecto, ya que el equipo de desarrollo disponía de uno con las capacidades necesarias para poner en funcionamiento los recursos software.

## 3.3. Costes

### 3.3.1. Costes humanos

Para calcular los costes humanos del proyecto, es necesario conocer el sueldo del equipo de desarrollo. Para tomar una referencia lo más realista posible, se ha desglosado el trabajo realizado en roles según las funciones que se han desempeñado y el tiempo que se ha dedicado a esas funciones. Además, se ha buscado el sueldo medio anual de cada rol y se han calculado el coste medio mensual con 14 pagas (12 pagas mensuales y 2 pagas extra), y el coste medio diario.

| **Rol** | **Salario medio anual** | **Salario medio mensual** | **Salario medio diario** |
|---|---|---|---|
| Business Analyst | 32417 € | 2316 € | 77 € |
| Solution Architect | 31700 € | 2264 € | 75 € |
| UX Designer | 26600 € | 1900 € | 63 € |
| Developer | 27138 € | 1938 € | 65 € |
| Tester | 26324 € | 1880 € | 63 € |
| Documentalist | 31165 € | 2226 € | 74 € |

Cuadro 3.1: Tabla de sueldos basados en roles

Una vez que ya disponemos del salario diario según los roles, procedemos a calcular los costes humanos en función del tiempo empleado en las funciones de cada rol.

| **Rol** | **Salario diario** | **Tiempo (días)** | **Coste/día** |
|---|---|---|---|
| Business Analyst | 3 | 77 € | 231 € |
| Solution Architect | 158 | 75 € | 11850 € |
| UX Designer | 5 | 63 € | 315 € |
| Developer | 34 | 65 € | 2210 € |
| Tester | 4 | 63 € | 252 € |
| Documentalist | 9 | 75 € | 675 € |
| **Total** | **213** | | **15533 €** |

Cuadro 3.2: Costes humanos del proyecto

### 3.3.2. Costes materiales

Para el cálculo de los costes materiales se han tenido en cuenta los siguientes elementos:

- *Hardware*: Elementos físicos que se han utilizado para el desarrollo del proyecto
- *Software*: Programas de los que se han hecho uso para la elaboración de la aplicación
- *Varios*: Costes varios de elementos necesarios para poder realizar el trabajo correctamente, tales como el gasto de luz o el coste de internet.

#### Hardware

Para calcular el coste del hardware, hemos contemplado el valor que queda por amortizar del ordenador que se ha utilizado para trabajar.

| Elemento | Valor | Vida útil | Tiempo de vida | Valor/año |
|---|---|---|---|---|
| Ordenador | 1726 € | 5 | 4 | 345 € |
| **Total** | | | | **345 €** |

Cuadro 3.3: Costes de hardware

#### Software

Como todo el software que se ha utilizado es de uso gratuito, el coste en este apartado es de 0 €.

| Software | Coste/día |
|---|---|
| Visual Studio 2019 Community | 0 € |
| Xamarin Forms | 0 € |
| Protégé | 0 € |
| Visual Studio Code | 0 € |
| LaTeXWorkshop | 0 € |
| MiKTeX | 0 € |
| **Total** | **0 €** |

Cuadro 3.4: Costes de software

#### Varios

En este apartado se han considerado los gastos de electricidad y conexión a Internet, que han sido indispensables para llevar a cabo el trabajo.

| **Software** | **Coste/mes** | **Coste/día** | **Coste total** |
|---|---|---|---|
| Luz | 57 € | 2 € | 430 € |
| Internet | 50 € | 2 € | 430 € |
| **Total** | | | **860 €** |

Cuadro 3.5: Costes varios

Tras calcular los diferentes costes materiales por separado, solo falta sumar sus valores para obtener el coste material total.

| **Elementos** | **Coste** |
|---|---|
| Hardware | 345 € |
| Software | 0 € |
| Varios | 860 € |
| **Total** | **1205 €** |

Cuadro 3.6: Costes varios

### 3.3.3. Costes totales

Calculados los costes humanos y materiales, podemos obtener el coste total del proyecto realizando la suma de estos.

| **Elementos** | **Coste** |
|---|---|
| Costes humanos | 15533 € |
| Costes materiales | 1205 € |
| **Total** | **16738 €** |

Cuadro 3.7: Coste total del proyecto

## 3.4. Evaluación y gestión de riesgos

### 3.4.1. ¿Qué es un riesgo?

Se considera **riesgo** en un proyecto de implantación de software cualquier eventualidad que pueda suponer una desviación del plan previsto y que posibilite el fracaso del proyecto.

### 3.4.2. Gestión de riesgos

La gestión de riesgos permite al equipo de desarrollo definir de forma lógica una serie de actividades para analizar los riesgos que se puedan presentar a lo largo del ciclo

de vida del proyecto, calcular su exposición y priorizarlos, de manera que se puedan establecer estrategias de control, resolución y supervisión de los mismos.

### 3.4.3. Identificación de riesgos

En primer lugar, se procederá a realizar una lista que identificará los posibles riesgos que puedan surgir a lo largo del proyecto:

- **Riesgos de planificación**: La planificación mal realizada puede retrasar enormemente el proyecto, pudiendo resultar en el fracaso del mismo. Posibles motivos para esto son una mala estimación de los tiempos de desarrollo, imposición de plazos por parte del cliente o una planificación optimista.

- **Riesgos de organización y gestión del proyecto**: Es posible que una parte del personal abandone el proyecto, lo que puede derivarse en el fracaso del proyecto si no se encuentra un sustituto adecuado rápidamente.

- **Riesgos de infraestructura hardware y software**: Es posible que surjan problemas con las herramientas, tales como incompatibilidad entre herramientas o incompatibilidad entre herramientas y dispositivos.

- **Riesgos de requisitos**: Es posible que los requisitos generen contratiempos, por ser añadidos constantemente o por modificar drásticamente lo previamente desarrollado.

- **Riesgos de diseño e implementación**: Es posible la falta de un diseño adecuado provoque obstáculos, ya sea por no realizar el diseño y pasar directamente a la implementación, porque el diseño esté demasiado simplificado para la complejidad del proyecto, o por tratar de implementar funciones no soportadas por las herramientas de trabajo.

### 3.4.4. Reducción de riesgos

En esta sección se van a describir las medidas que el equipo ha considerado oportunas para minimizar la aparición de los riesgos previamente indicados.

- **Riesgos de planificación**: Se realizará una planificación objetiva, considerando que es susceptible a tener problemas, que en el caso de darse, se estudiarán para modificar la planificación existente de manera que se puedan sortear minimizando su efecto en el desarrollo del proyecto.

- **Riesgos de organización y gestión del proyecto**: Es posible que una parte del personal abandone el proyecto, lo que puede derivarse en el fracaso del proyecto si no se encuentra un sustituto adecuado rápidamente.

- **Riesgos de infraestructura hardware y software**: Se hará un estudio previo de las herramientas para comprobar que es posible trabajar con ellas sin problema alguno. También se realizarán copias de seguridad periódicamente, de manera que de darse un riesgo imprevisible como que el dispositivo de almacenamiento sufra un error y la información sea irrecuperable, haya alguna manera de obtener una versión anterior del proyecto, lo más actualizada posible, desde la que se pueda partir.
- **Riesgos de requisitos**: Se realizará una recolección exhaustiva de requisitos del proyecto al comienzo, para evitar en la medida de lo posible la aparición de requisitos extra durante el desarrollo del proyecto. Todos los requisitos quedarán documentados.
- **Riesgos de diseño e implementación**: En cada etapa del desarrollo, se comprobará de forma periódica si el diseño puede obviar alguno de los requisitos establecidos.

# Parte II

# Desarrollo

# CAPÍTULO 4

# Requisitos del Sistema

## 4.1. Catálogo de requisitos

Como dice Távora [58], un requisito es una cláusula que debe cumplir una aplicación de manera que se logre un objetivo o se solvente algún problema.

En la presente sección se procederá a realizar un análisis de requisitos del sistema, que recoge y describe el conjunto de requisitos específicos del sistema que se va a desarrollar. Posteriormente, se describirán los casos de uso en el próximo capítulo.

Para catalogar los requisitos del sistema, se hará una diferenciación entre *requisitos funcionales*, que son aquellos que detallan la funcionalidad del sistema, y *requisitos no funcionales*, que refieren a otros aspectos del software que deben ser satisfechos.

### 4.1.1. Requisitos funcionales

Citando a Wiegers y Beatty [59], *"los requisitos funcionales detallan los comportamientos observables que el sistema mostrará bajo ciertas condiciones y las acciones que el sistema dejará tomar a los usuarios."* A continuación se muestran los requisitos funcionales de este proyecto

### 4.1.2. Requisitos no funcionales

Los usuarios pueden tener ideas sobre cómo funcionará la aplicación, tales como la fiabilidad, la rápidez de ejecución, la facilidad de uso, etc. Estos aspectos quedan definidos por los *requisitos no funcionales*, como expresan Stellman y Greene [60].

Para la declaración de requisitos no funcionales, se establecerán como base los requisitos indicados en las normas *IEEE Std. 830* e *ISO/IEC 25010 (SQuaRE)*:

| *Código* | *Requisito* |
|---|---|
| **RF-01** | El sistema debe permitir al usuario seleccionar un juego del conjunto de juegos incluidos. |
| **RF-02** | El sistema debe permitir al usuario crear paso a paso un personaje para el juego seleccionado. |
| **RF-03** | El sistema debe adaptar el proceso de creación de personajes al juego seleccionado, sin importar su complejidad. |
| **RF-04** | El sistema debe permitir la visualización de la información de un personaje creado por el usuario. |
| **RF-05** | El sistema debe permitir la modificación parcial o total de la información de un personaje creado por el usuario. |
| **RF-06** | El sistema debe permitir la eliminación de un personaje creado por el usuario |
| **RF-07** | El sistema debe permitir realizar cálculos con la información de un personaje creado por el usuario. |

Cuadro 4.1: Tabla de requisitos funcionales

- **Adecuación funcional**: La aplicación debe cumplir con todos los requisitos necesarios, de manera que sea completo y correcto funcionalmente.
- **Seguridad**: El sistema no requiere asegurar la información que procesa, debido a que no contiene información sensible del usuario en ningún momento, ni realiza conexión externa alguna para obtener información.
- **Compatibilidad**: La aplicación deberá ser compatible con los ficheros que contienen la información de los juegos que formarán parte del sistema.
- **Usabilidad**: El sistema debe disponer de una interfaz de usuario intuitiva y fácil de manejar, de manera que pueda ser utilizado por usuarios sin conocimientos técnicos ni avanzados de informática. La curva de aprendizaje deberá ser lo más reducida posible, de manera que personas de cualquier ámbito puedan hacer uso del mismo.
- **Fiabilidad**: La aplicación deberá estar libre de errores que influyan negativamente en su uso normal. Debido a que la aplicación depende de información incluida por terceros, será necesario comprobar que dicha información es compatible con la aplicación.
- **Eficiencia**: El sistema debe evitar en la medida de lo posible el uso de información redundante para poder asegurar su funcionamiento cuando se introduzcan juegos de alta complejidad que requieran un elevado uso de recursos.

- **Mantenibilidad**: Este apartado representa la capacidad del producto software para ser modificado efectiva y eficientemente. Esto será posible debido al desarrollo de código limpio y bien documentado, al diseño y la implementación modular del mismo. Se plantea el uso de patrones de arquitectura de software, tales como **MVVM**.

- **Portabilidad**: El sistema está diseñado para su uso en dispositivos móviles, aunque no se descarta una futura ampliación para introducirlo en otro tipo de dispositivos. La implementación está realizada únicamente para sistemas *Android*, ya que no se dispone de las herramientas necesarias para el despliegue en *Mac OS*.

## 4.2. Alternativas de Solución

En esta sección, se ofrece un estudio del arte de las diferentes alternativas tecnológicas que permitan satisfacer los requerimientos del sistema, para optar por una de las opciones planteadas, que será dispuesta como base para el software a desarrollar.

Con este motivo, hemos optado por recoger algunas de las tecnologías existentes para realizar desarrollo de aplicaciones móviles.

### 4.2.1. Entorno de desarrollo integrado

Se conoce por *Entorno de Desarrollo Integrado* o *IDE* a las aplicaciones visuales que se utilizan para la elaboración de aplicaciones a partir de componentes, según explican Fuentes, Troya y Vallecillo. [61] Estas aplicaciones suelen disponer de ciertos elementos comunes, tales como:

- Colecciones que muestran los componentes disponibles mediante iconos.

- Un elemento central en el que se posicionan los componentes y se interrelacionan entre sí.

- Editores específicos para personalizar los componentes.

- Buscadores para localizar componentes.

- Directorios de componentes.

- Capacidad de desarrollo de nuevos componentes mediante el uso de los componentes ya disponibles.

- Acceso a herramientas de control y gestión de proyectos.

A continuación, mostramos los *IDEs* que se han tomado en consideración para el desarrollo del presente proyecto.

### Android Studio

Citando la página oficial, *"Android Studio es el entorno de desarrollo integrado (*IDE*) oficial para el desarrollo de apps para Android, basado en* IntelliJ IDEA*."* [62] Disponible para las plataformas *Windows*, *macOS* y *GNU/Linux*. Basado en el lenguanje *Java*, no tiene herramientas nativas para trabajar directamente con *RDF* y *OWL*. Para suplir este obstáculo, haríamos uso del framework libre *Apache Jena*, cuya API permite trabajar con RDF, consiguiendo vincular el desarrollo en aplicaciones móviles con el uso de ontologías.


Figura 4.1: Logo de *Android Studio*


Figura 4.2: Logo de *Apache Jena*

### React Native

Según indica Wikipedia [63], *"React Native es un framework para el desarrollo de aplicaciones móviles de código abierto desarrollado por Facebook. Se utiliza para desarrollar aplicaciones para Android, iOS, Web y U permitiendo a los desarrolladores usar React con funcionalidades nativas de las plataformas."* Al igual que *Android Studio*, React Native no tiene herramientas nativas para el desarrollo de ontologías, de manera que haríamos uso de bibliotecas tales como *rdflib.js* para poder proceder al tratamiento de las ontologías.

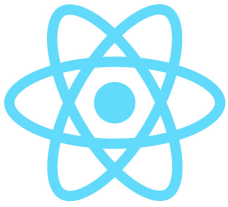

Figura 4.3: Logo de *React Native*


Figura 4.4: Logo de *rdflib.js*

### Xamarin

Microsoft define Xamarin como una plataforma de código abierto para compilar aplicaciones modernas para *iOS*, *Android* y *Windows* con *.NET* [26]. Xamarin es una capa de abstracción que administra la comunicación de código compartido con el código de plataforma subyacente. Xamarin dispone de varias bibliotecas que permiten trabajar con el formato *RDF*, como pueden ser **dotNetRDF**, **FSharp.RDF**, **LITEQRDF** o **RdfMapperNet**, pero muy pocas trabajan con el formato *OWL*, lo que hace que resulte fácil realizar un filtro de entre todas las posibilidades existentes, y optemos por la biblioteca ***RDFSharp***, creada por Marco de Salvo, que permite trabajar con ambos formatos, necesarios para el desarrollo de nuestro modelo ontológico.


Figura 4.5: Logo de *Xamarin*

## 4.3. Solución propuesta

Tras considerar las opciones planteadas en el apartado anterior, se ha considerado descartar *Android Studio* en primer lugar, ya que sólo permite el desarrollo en *Android*, mientras que las otras soluciones permiten realizar el desarrollo en varias plataformas.

Una vez desechada una de las opciones, se ha comprobado que las soluciones restantes son compatibles con el proyecto, y prácticamente generan el mismo resultado. Por ello, en vez de considerar las plataformas, se ha realizado una comparación en función al lenguaje de programación con el que se trabaja en cada una de ellas, que son **JavaScript** en *React Native*, y **C#** para *Xamarin*. Esta comparativa se realizará en forma de tabla.

Cuadro 4.2: Comparativa entre *JavaScript* y *C#*

| | ***JavaScript*** | ***C#*** |
|---|---|---|
| **Tipo de Lenguaje** | Scripting | Orientado a Objetos |
| **Tipado** | Débil | Fuerte |
| **Detección de errores** | Ejecución | Compilación y ejecución |
| **Compilación** | No | Sí |
| **Mantenibilidad** | Complejo | Sencillo |
| **Soporte de IDE** | No | Microsoft Visual Studio |
| **Sintaxis** | OBSL | OOP |

En la comparativa se puede observar que JavaScript es un lenguaje de scripting débilmente tipado que no requiere ser compilado, pero resulta difícil de mantener en sistemas complejos. Por otro lado, C# es un lenguaje orientado a objetos fuertemente tipado que requiere ser compilado, que permite una mayor facilidad a la hora de mantener el código en sistemas de alta complejidad.

De entre estas dos posibilidades, el equipo de desarrollo ha tomado la decisión de que la mejor herramienta para la elaboración del presente proyecto es ***Xamarin*** con la biblioteca ***RDFSharp***, ya que el C# es un lenguaje orientado a objeto, y este paradigma es bastante similar a la estructura del modelo ontológico, que será uno de los pilares de la aplicación. Además, que tenga una mantenibilidad sencilla de comprender se compatibiliza con la idea de que cualquier usuario con los conocimientos suficientes pueda realizar ampliaciones en su funcionalidad.

# CAPÍTULO 5

# Análisis del Sistema

## 5.1. Modelo Conceptual

### 5.1.1. Introducción

Un modelo conceptual es una representación clara y precisa de las características estructurales y comportamentales de un sistema utilizando un formato predefinido, de acuerdo con Robinson et al. [64], es decir, un patrón que permite conocer cómo se comporta un sistema concreto y como está dispuesto.

Uno de los principales objetivos de este proyecto consiste en elaborar un banco de modelos que permita al usuario acceder a los juegos cuyo modelo esté incluido, de manera que pueda interactuar con información de diferentes *RPGs* sin la necesidad de cambiar de aplicación. Esto resulta sencillo para juegos con mecánicas similares, ya que varía la información del juego, pero la estructura del proceso de creación se mantiene intacta.

El problema surge cuando tratamos de utilizar juegos con mecánicas diferentes, pues el proceso de creación debe modificarse parcial o completamente, pero a su vez debe ofrecer toda la información requerida por el usuario para proceder a la creación de personajes. Además, no todos los juegos tienen el mismo grado de complejidad, puesto que hay juegos que requieren multitud de elementos para la creación de personajes, mientras otros juegos sólo necesitan el nombre del personaje y la imaginación del usuario.

Cada juego tiene su propio üniverso, en el que se modela un mundo con unas reglas concretas. Para que la aplicación sea capaz de procesar todos los universos, representados mediante ontologías, se hace necesario *generalizar* la estructura de estas, de manera que el sistema de acceso a la información sea idéntico para todas las ontologías, pero cada una de ellas pueda ser explotada de forma completa.

Así pues, en vez de realizar un modelo conceptual sobre una ontología específica, se va a formular un modelo conceptual **genérico**, que permita identificar los diferentes tipos de elementos de los que puede disponer cualquier ontología que vaya a formar parte del banco de datos de la aplicación.

Una vez aclarado que el modelo conceptual será genérico, cabe destacar que para realizar una generalización, es necesario trabajar con un elemento de pruebas como ejemplo, que permita al equipo de desarrollo abstraer los aspectos generales del juego y obtener un modelo eficaz que pueda ser implementado.

Durante todo el proyecto, se utilizará como juego de ejemplo **Ánima: Beyond Fantasy** (al que llamaremos Ánima para abreviar). Esto se debe a que tiene un nivel de complejidad alto en comparación con otros juegos, lo que permite a los desarrolladores considerar que, si el modelo es útil para *Ánima*, entonces es igual de válido para juegos de menor complejidad.

### 5.1.2. UML

Actualmente hay una gran variedad de lenguajes para describir arquitecturas software, pero no hay consenso para decidir qué conjunto de símbolos y sistema de vista debe ser adoptado de manera general. Según indican Kaur y Singh [65], el *lenguaje de modelado unificado o* ***UML*** posibilita a los desarrolladores describir, mostrar y documentar sistemas completos, facilitando su colaboración con perspectivas diferentes, gracias a la disposición de una herramienta que les ayuda a alcanzar una base común para comunicarse.

Teniendo en consideración lo anterior, el equipo de desarrollo ha considerado oportuno realizar el modelo conceptual utilizando *UML* como base. Para ello, se utilizará el mismo modelo de transformación de elementos *OWL* a *UML* que aplica Ortega en su trabajo *Transformación de ontologías OWL a UML para la indexación y recuperación mediante esquema basado en grafos* [66], ya que facilita una conversión clara de los elementos pertenecientes a la ontología a *UML* y viceversa, y asímismo, se puede hacer uso de esta para desarrollar un esquema *UML* para realizar su conversión a formato *OWL 2*.

### 5.1.3. Elementos

El modelo conceptual desarrollado, que se muestra en la figura 5.1, está basado en los componentes que conforman una ontología y sus propiedades:

- ***Individual***: Citando a Bock, Fokoue, Haase et al. [67], un individuo es la representación de un objeto real en el dominio.
- ***Class***: Una clase es un conjunto de individuos que comparten las mismas propiedades y/o comportamientos.

- ***Object Property***: Una propiedad objeto es una relación que conecta pares de individuos.

- ***Datatype Property***: Una propiedad de datos asocia un valor con un individuo.

- ***Annotation Property***: Una propiedad de anotación provee información extra a una ontología, un axioma o un *Identificador de Recursos Internacionalizado (**IRI**)*.

Como las anotaciones solo pueden relacionarse con ontologías completas, axiomas o *IRIs*, resulta imposible asignarlas a cualquier clase, por lo que se ha tomado la decisión de incluir una clase llamada **ClassDefinition**, en la que cada individuo está directamente relacionado con una clase de la ontología, y que dispone de todas las anotaciones necesarias para que la información de la clase a la que hace referencia esté completa.

### 5.1.4. Relaciones

Los elementos del modelo conforman una parte importante, pero éste no quedaría completo sin tratar las relaciones que hay entre ellos. Para tratarlas vamos a separarlas en función de los elementos que relacionan.

#### Individual

- Un individuo puede o no tener una o varias relaciones de datos.

- Un individuo puede o no tener una o varias relaciones de objeto con otro u otros individuos.

- Un individuo tiene una o varias propiedades de anotación.

#### Class

- Una clase puede o no estar formada por un conjunto de individuos.

- Una clase puede o no estar relacionada con una o varias clases superiores en la jerarquía.

- Una clase está relacionada con una definición de clase.

#### ClassDefinition

- Una definición de clase tiene una o varias propiedades de anotación.

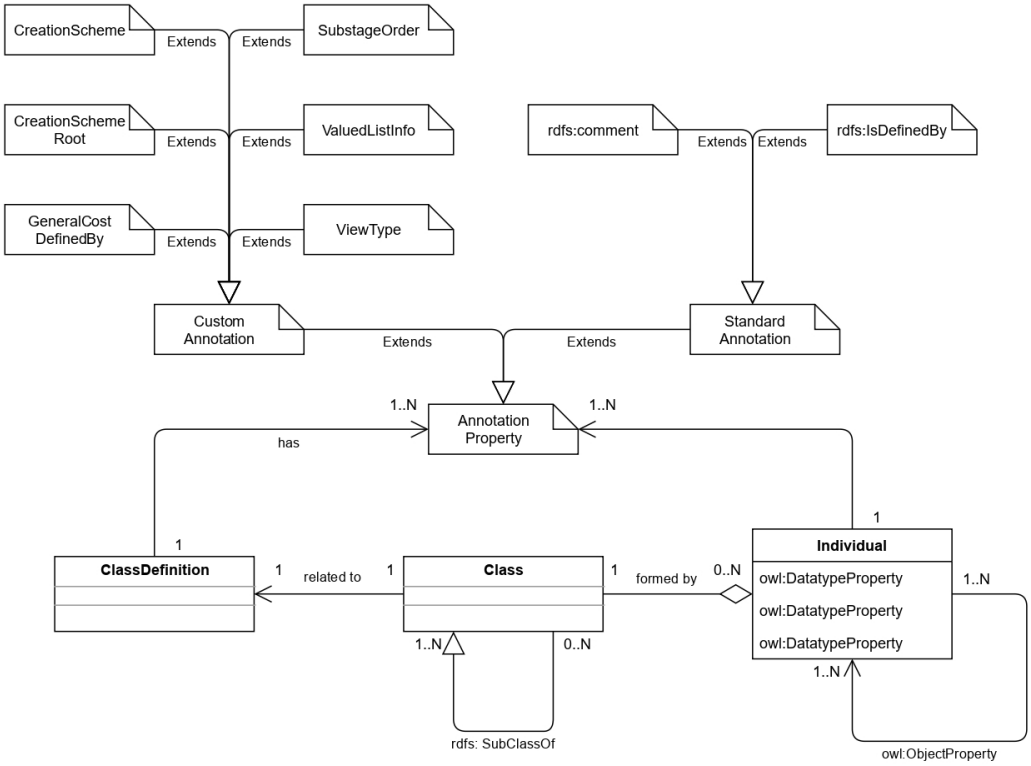


Figura 5.1: Modelo conceptual

## 5.2. Modelo de Casos de Uso

Ahora que se dispone de una estructura clara para el diseño de ontologías del sistema, hay que explicar cómo se resuelven los diferentes requisitos funcionales comentados en el apartado 4.1.1. Por esa razón, en este apartado se mostrarán los diferentes *casos de uso* del sistema, que, como plantea Cockburn [68], describen las diferentes interacciones posibles entre los actores y el sistema asociadas a una meta concreta.

### 5.2.1. Actores

En este sistema no se aprecia diferencia alguna entre usuarios, por lo que todos ellos se consideran como un único agente externo, disponiendo todos ellos de los mismos privilegios.

| ***Actor*** | ***Descripción*** |
|---|---|
| **Usuario** | Es la persona que interacciona con la aplicación, haciendo uso de las funciones que esta posee. |

Cuadro 5.1: Tabla de actores

### 5.2.2. Descripción de casos de uso

Inicialmente, se va a describir un caso de uso de alto nivel que representa todos los requisitos del sistema que requieren algún tipo de interacción por parte del usuario.

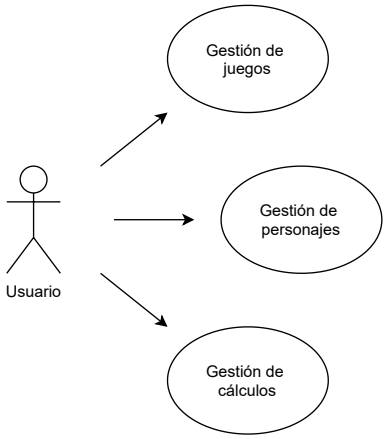


Figura 5.2: Diagrama de Casos de Uso del sistema

Como en la mayoría de casos de uso es necesario conocer el juego que se va a utilizar, se va a considerar que en los casos que no estén directamente relacionados con la gestión de juegos, el primer paso es *Seleccionar juego*, para simplificar los diagramas y facilitar su comprensión.

### 5.2.3. Gestión de juegos

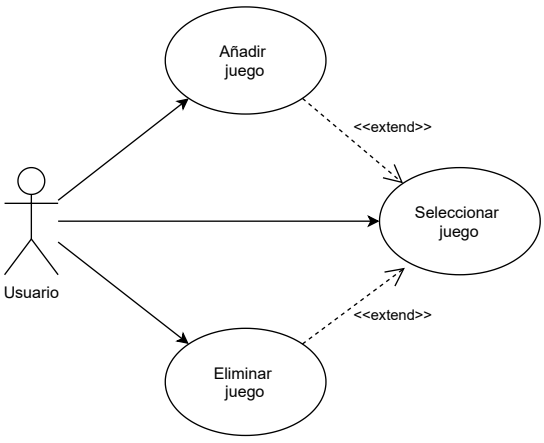


Figura 5.3: Diagrama de Casos de Uso: Gestión de juegos

**Caso de uso: Añadir juego**

- **Descripción**: El usuario añade un juego o una versión de un juego a la aplicación.
- **Precondiciones**: El usuario debe disponer del fichero de información del juego en el dispositivo que contiene la aplicación.
- **Postcondiciones**: La aplicación tiene acceso a un nuevo juego.

***Identificación de escenarios***

**Escenario principal**

1. El usuario desea introducir un nuevo juego en la aplicación.
2. El usuario pulsa el botón de *Añadir juego*.
3. El usuario selecciona la opción *Juego nuevo*.
4. El usuario introduce el nombre del juego.
5. El usuario indica el fichero de información del juego.
6. El usuario pulsa el botón de *Aceptar*.
7. El sistema crea la estructura de carpetas para el nuevo juego.
8. El sistema copia el fichero en su nueva ubicación, de manera que ya está listo para su uso en el sistema.

**Escenario alternativo: Añadir nueva versión de un juego ya existente**

1. El usuario desea introducir un nuevo juego en la aplicación.
2. El usuario pulsa el botón de *Añadir juego*.
3. El usuario selecciona la opción *Actualizar juego*.
4. El usuario indica el directorio en el que debe ubicarse el juego que desea incluir.
5. El usuario indica el fichero de información del juego.
6. El usuario pulsa el botón de *Aceptar*.
7. El sistema copia el fichero en su nueva ubicación, de manera que ya está listo para su uso en el sistema.

**Caso de uso: Seleccionar juego**

- **Descripción**: El usuario elige un juego disponible en la aplicación.
- **Precondiciones**: El usuario debe disponer de algún juego en la aplicación.
- **Postcondiciones**: La aplicación toma el juego y la versión indicados como juego y versión activos hasta que se realice una nueva selección.

***Identificación de escenarios***

**Escenario principal**

1. El usuario desea seleccionar un juego de la aplicación.
2. El usuario pulsa el botón de *Seleccionar juego*.
3. El sistema muestra un listado con los juegos disponibles.
4. El usuario selecciona una de las opciones mostradas.
5. El sistema muestra todas las versiones disponibles del juego seleccionado.
6. El usuario selecciona una de las opciones mostradas.
7. El usuario pulsa el botón de *Aceptar*.
8. El sistema crea la estructura de carpetas para el nuevo juego.
9. El sistema copia el fichero en su nueva ubicación, de manera que ya está listo para su uso en el sistema.

**Caso de uso: Eliminar juego**

- **Descripción**: El usuario elimina un juego disponible de la aplicación.
- **Precondiciones**: El usuario debe disponer de algún juego en la aplicación.
- **Postcondiciones**: La aplicación elimina toda la información relacionada con el juego en cuestión.

***Identificación de escenarios***

**Escenario principal**

1. El usuario desea eliminar un juego de la aplicación.
2. El usuario pulsa el botón de *Eliminar juego*.
3. El usuario selecciona la opción de *Eliminar juego completo*.
4. El sistema muestra un listado con los juegos disponibles.
5. El usuario selecciona una de las opciones mostradas.
6. El sistema realiza una pregunta de confirmación.
7. El usuario pulsa la opción *Eliminar*.
8. El sistema elimina todos los elementos relacionados con el juego seleccionado.

**Escenario alternativo: Eliminar una versión de un juego sin eliminar el juego completo**

1. El usuario desea eliminar un juego de la aplicación.
2. El usuario pulsa el botón de *Eliminar juego*.
3. El usuario **no** selecciona la opción de *Eliminar juego completo*.
4. El sistema muestra un listado con los juegos disponibles.
5. El usuario selecciona una de las opciones mostradas.
6. El sistema muestra todas las versiones disponibles del juego seleccionado.
7. El usuario selecciona una de las opciones mostradas.
8. El usuario pulsa el botón *Aceptar*.
9. El sistema realiza una pregunta de confirmación.
10. El usuario pulsa la opción *Eliminar*.
11. El sistema elimina la versión indicada del juego seleccionado.

### 5.2.4. Gestión de personajes

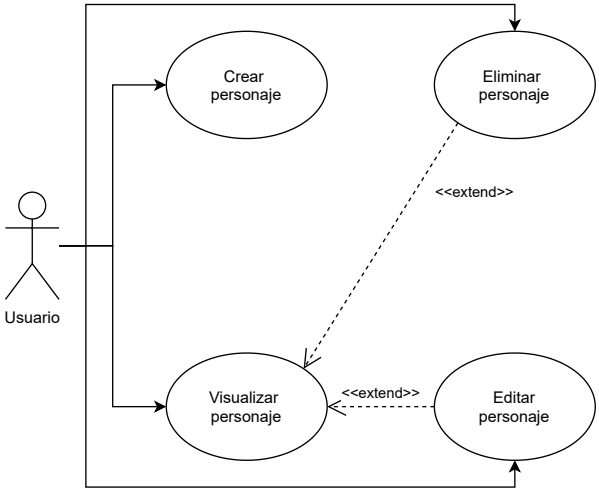


Figura 5.4: Diagrama de Casos de Uso: Gestión de personajes

**Caso de uso: Crear personaje**

- **Descripción**: El usuario crea un personaje siguiendo las normas del juego y la versión activos.
- **Precondiciones**: El usuario debe haber seleccionado el juego deseado previamente.
- **Postcondiciones**: La aplicación genera un fichero con la información del personaje y lo almacena en la carpeta de personajes correspondiente al juego activo.

***Identificación de escenarios***

**Escenario principal**

1. El usuario desea crear un personaje del juego activo en la aplicación.
2. El usuario pulsa el botón de *Crear personaje*.
3. El usuario introduce el nombre del personaje.
4. El sistema muestra las diversas etapas de creación del personaje correspondiente del juego, presentadas a modo de carrusel, de manera que permite al usuario avanzar y retroceder de etapa según lo necesite.
5. El usuario configura el personaje a su medida.

6. El usuario presiona el botón *Finalizar* una vez ha terminado de configurar el personaje.
7. El sistema genera el fichero del personaje y lo almacena en la carpeta de personajes del juego correspondiente.

**Caso de uso: Visualizar personaje**

- **Descripción**: El usuario visualiza un personaje del juego activos.
- **Precondiciones**: El personaje debe haber sido creado previamente.
- **Postcondiciones**: La aplicación muestra la información del personaje.

***Identificación de escenarios***

**Escenario principal**

1. El usuario desea visualizar la información de un personaje del juego activo en la aplicación.
2. El usuario pulsa el botón de *Visualizar personaje*.
3. El sistema muestra el listado de personajes del juego activo.
4. El usuario selecciona una de las opciones mostradas.
5. El sistema muestra la información del usuario, separada por etapas a modo de carrusel, de manera que permite al usuario navegar entre etapas según lo necesite.

**Escenario alternativo: Editar información del personaje**

1. Este escenario se describe en el caso de uso *Editar personaje*.

**Escenario alternativo: Eliminar personaje**

1. Este escenario se describe en el caso de uso *Eliminar personaje*.

**Caso de uso: Editar personaje**

- **Descripción**: El usuario modifica información de un personaje del juego activo.
- **Precondiciones**: El personaje debe haber sido creado previamente.
- **Postcondiciones**: La aplicación sobreescribe la información del personaje.

***Identificación de escenarios***

**Escenario principal**

1. El usuario desea visualizar la información de un personaje del juego activo en la aplicación.
2. El usuario pulsa el botón de *Visualizar personaje*.
3. El sistema muestra el listado de personajes del juego activo.
4. El usuario selecciona una de las opciones mostradas.
5. El sistema muestra la información del usuario, separada por etapas a modo de carrusel, de manera que permite al usuario navegar entre etapas según lo necesite.
6. El usuario pulsa el botón de *Editar información de etapa*.
7. El sistema muestra la vista de edición de la etapa en cuestión.
8. El usuario altera en parte o en su totalidad el contenido de la vista.
9. El usuario pulsa el botón *Guardar cambios*
10. El sistema realiza una pregunta de seguridad.
11. El usuario pulsa el botón *Guardar*.
12. El sistema sobreescribe el fichero del personaje con los nuevos valores.

**Caso de uso: Eliminar personaje**

- **Descripción**: El usuario elimina un personaje del juego activo.
- **Precondiciones**: El personaje debe haber sido creado previamente.
- **Postcondiciones**: La aplicación sobreescribe la información del personaje.

***Identificación de escenarios***

**Escenario principal**

1. El usuario desea visualizar la información de un personaje del juego activo en la aplicación.
2. El usuario pulsa el botón de *Visualizar personaje*.
3. El sistema muestra el listado de personajes del juego activo.
4. El usuario selecciona una de las opciones mostradas.

5. El sistema muestra la información del usuario, separada por etapas a modo de carrusel, de manera que permite al usuario navegar entre etapas según lo necesite.
6. El usuario pulsa el botón de *Eliminar personaje*.
7. El sistema realiza una pregunta de seguridad.
8. El usuario pulsa el botón *Eliminar*.
9. El sistema elimina el fichero del personaje.

### 5.2.5. Gestión de habilidades

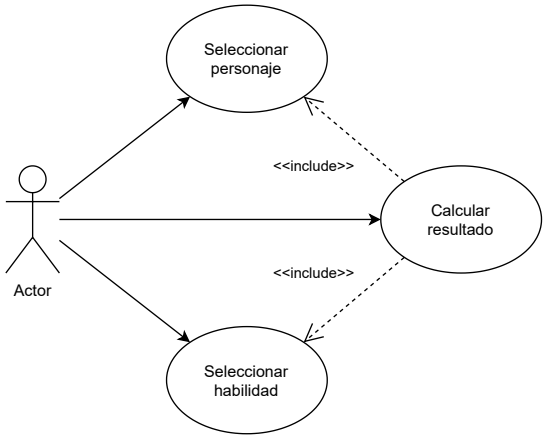


Figura 5.5: Diagrama de Casos de Uso: Gestión de habilidades

**Caso de uso: Calcular habilidad**

- **Descripción**: El usuario conocer el resultado de una habilidad aplicando los modificadores correspondientes al resultado del lanzamiento de dados.
- **Precondiciones**: El personaje debe haber sido creado previamente.
- **Postcondiciones**: La aplicación calcula el valor final de la habilidad aplicando los modificadores almacenados en el fichero del personaje.

***Identificación de escenarios***

**Escenario principal**

1. El usuario desea calcular el resultado de una habilidad de un personaje.
2. El usuario pulsa el botón de *Calcular habilidad de personaje*.
3. El sistema muestra la pantalla de cálculo de habilidades.

4. El usuario pulsa el botón de *Seleccionar personaje*.
5. El sistema muestra el listado de personajes del juego activo.
6. El usuario selecciona una de las opciones mostradas.
7. El usuario pulsa el botón de *Seleccionar habilidad*.
8. El sistema muestra un listado con las habilidades disponibles del personaje.
9. El usuario selecciona una de las opciones mostradas.
10. El usuario introduce el valor del lanzamiento de dados.
11. El usuario pulsa el botón de *Calcular*.
12. El sistema devuelve el resultado del cálculo.

## 5.3. Modelo de Comportamiento

Después de describir los diferentes casos de uso de los requisitos del sistema, el próximo paso consiste en elaborar el *Modelo de Comportamiento*. Este modelo está comprendido por diagramas de secuencia del sistema, identificando operaciones y servicios del mismo.

Como varios diagramas de los mostrados en el apartado anterior tienen comportamientos muy parecidos, no se mostrarán todos los diagramas de secuencia, sino un conjunto reducido de éstos.

Otro detalle que cabe destacar es que no se puede concretar de qué manera interactúa el usuario con la aplicación en las vistas que dependen de la información del juego. Por tanto se hará uso de la función *Interactuar()* para representar la interacción del usuario con el sistema en estos casos. Una vez finalizada la función, se considerará que el usuario ha realizado todas las acciones que haya estimado oportunas.

### 5.3.1. Pantalla de selección

Para mostrar una lista de selección, el sistema busca todos los elementos de la lista y la muestra. Una vez el usuario dispone de la lista de selección, el usuario puede elegir una opción (o varias, en función del elemento que se seleccione), quedando ésta marcada en la pantalla de selección.

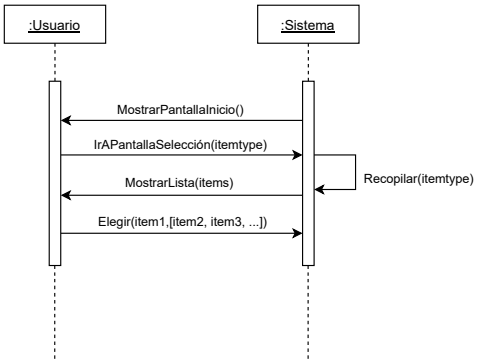


Figura 5.6: Modelo de Comportamiento:Pantalla de selección

### 5.3.2. Crear personaje

El proceso de creación de personajes puede considerarse el más complejo, pues debe ser dinámico para adaptarse a cualquiera de los juegos introducidos en la aplicación.

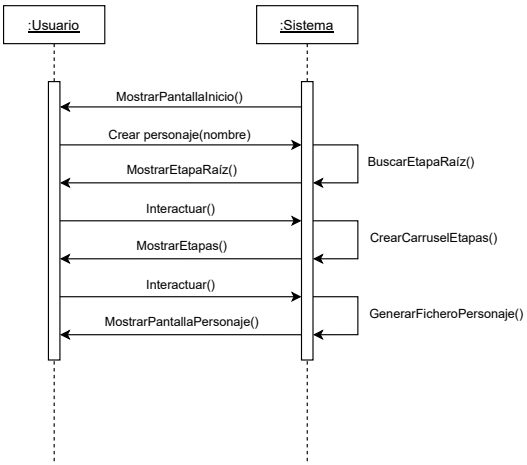


Figura 5.7: Modelo de Comportamiento: Crear personaje

### 5.3.3. Editar personaje

Para editar un personaje, será necesario haber accedido a la visualización del mismo, para que el usuario sea consciente de qué personaje es el que está editando.

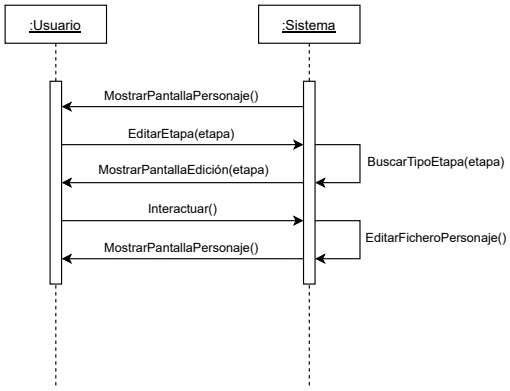


Figura 5.8: Modelo de Comportamiento: Editar personaje

### 5.3.4. Eliminar personaje

Al igual que en el caso anterior, será necesario haber accedido a la visualización del personaje antes de eliminarlo, de manera que el usuario sea consciente de qué personaje es el que está eliminando, ya que una vez hecho, no se podrá recuperar el fichero.

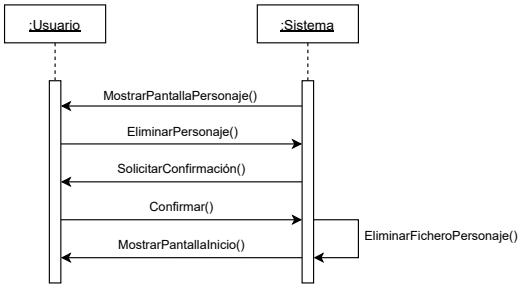


Figura 5.9: Modelo de Comportamiento: Eliminar personaje

### 5.3.5. Cálculo de habilidades

Para realizar el cálculo de habilidades, el usuario tendrá que indicar la habilidad que desea calcular y qué personaje la realiza para aplicar los valores correctos.

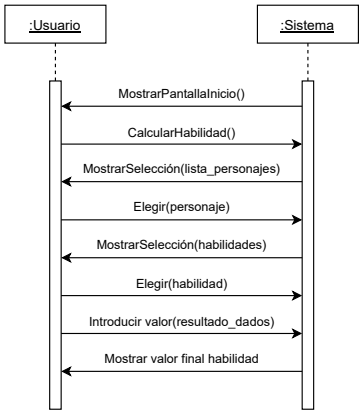


Figura 5.10: Modelo de Comportamiento: Cálculo de habilidades

## 5.4. Modelo de Interfaz de Usuario

CAPÍTULO 6

# Diseño del Sistema

## 6.1. Arquitectura del Sistema

Esta sección tratará de describir la arquitectura física y lógica del sistema. Para ello se abordará cómo está estructurado, su funcionamiento y la forma de interactuar entre los diferentes componentes del proyecto.

### 6.1.1. Arquitectura Física

El apartado de arquitectura física del sistema hace referencia a los componentes, tanto hardware como software, necesarios para poder lanzar la aplicación resultante del proyecto. Este proyecto no está únicamente orientado a los usuarios de la aplicación, sino que también está orientado a los desarrolladores de *RPGs*, dándoles acceso a una herramienta que sólo requiere la elaboración del fichero de información del juego, reduciendo los tiempos de producción del generador de personajes, de manera que los usuarios puedan disfrutar lo antes posible de un generador de personajes para un juego parcial o totalmente desconocido, lo que hace a su vez que la curva de aprendizaje para crear personajes de ese juego se reduzca considerablemente.

#### Requisitos para usuarios

El único requerimiento de esta aplicación es disponer de un dispositivo móvil que tenga *Android 9.0 (**Pie**)* o superior como sistema operativo. ***No se requiere conexión a Internet para acceder a la aplicación.***

### Requisitos para desarrolladores

En lo referente a desarrolladores de *RPGs*, es necesario tener instalado ***Protégé*** o una aplicación similar para poder crear la ontología del juego que deseen introducir en la aplicación. En caso de utilizar la versión de escritorio de *Protégé*, indicar que requiere al menos una versión de ***Java Runtime Environment***.

## 6.1.2. Arquitectura Lógica

En el apartado 2.3.2 el equipo de desarrollo indicó que ha tomado como método para elaborar la estructura del sistema el patrón ***MVVM***. Este patrón de arquitectura de software permite diferenciar claramente la lógica de una aplicación de su interfaz de usuario, lo que facilita el desarrollo, la elaboración de pruebas y la mantenibilidad del sistema, además de la reutilización del código y la colaboración entre desarrolladores y diseñadores [14].

### Ventajas

Las ventajas de utilizar el patrón *MVVM* son las siguientes:

- El modelo de vista permite que no sea necesario realizar ediciones masivas del código del modelo.
- Se pueden realizar pruebas para los modelos y los modelos de vista independientes a la vista.
- Las vistas pueden rediseñarse sin modificar la lógica de negocio de la aplicación.
- Permite independencia entre desarrolladores y diseñadores.

### Elementos

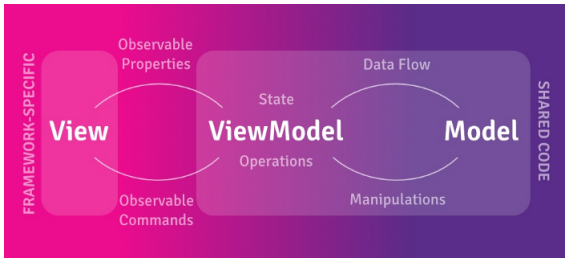


Figura 6.1: Estructura del patrón *MVVM*. Extraído de *medium.com* [69]

*MVVM* divide la estructura de un sistema software en tres tipos de elementos básicos, explicados de manera sencilla por Lou [70]:

- ***Modelo***: El modelo contiene los datos y estados de la aplicación, ya sean elementos simples o clases con estructuras complejas. Puede ser que no contenga la información de manera directa, pero que pueda recuperarla de manera remota.
- ***Modelo de vista***: El modelo de vista tiene como objetivo dirigir los diferentes estados de la vista, ya sea mediante el envío de operaciones y/o datos, o bien gestionando la lógica de la vista y su comportamiento.
- ***Vista***: El componente de vista muestra la interfaz de la aplicación, cuyo desarrollo está orientado más orientado a diseñadores que a programadores.

## 6.2. Diseño Lógico de Datos

### 6.2.1. Introducción

En esta sección se va a exponer de forma detallada el proceso de diseño de la ontología que será desarrollada para el proyecto. Con este objetivo, se realizará un estudio del diagrama conceptual que se muestra en el apartado 5.1. Este proceso de transformar el esquema del modelo conceptual a una estructura lógica que permita almacenar los datos de la aplicación de la manera adecuada, se conoce como *diseño lógico*.

Antes de abordar el proceso, cabe destacar que para poder realizar una ontología que permita comprobar que el proyecto puede trabajar con *RPGs* de todo tipo de complejidad, es necesario que el juego que se utilice como ejemplo sea lo más completo posible, de manera que juegos de igual o menor complejidad puedan ser desarrollados y utilizados asegurando la fiabilidad y completitud de la aplicación.

Por consiguiente, y tras realizar búsquedas de juegos de rol en base a su complejidad, el equipo de desarrollo ha tomado la decisión de utilizar como juego de ejemplo ***Ánima: Beyond Fantasy***. *Ánima* es un juego de rol de mesa que transcurre en un mundo de magia y fantasía, con un sistema bastante detallado y una complejidad muy elevada, hasta el punto de que hay jugadores que lo evitan debido a la larga duración del proceso de creación de personaje, por lo que hace de este juego un caso perfecto para demostrar la eficacia de este proyecto.

### 6.2.2. Metodología

Antes de proceder al desarrollo de la ontología de *Ánima*, se ha considerado fundamental plantear que es imposible abordar una fuente de información tan compleja como un *RPG* directamente. En consecuencia, el equipo de desarrollo ha procedido a modular el diseño de la ontología, de modo que se aborden los diferentes elementos de la ontología por separado, y poco a poco se vayan acoplando hasta que se obtenga el resultado final.

### Individuos

Como se ha indicado previamente en el apartado 5.1.3 de la sección 5.1, un individuo es la *la representacion de un objeto real en el dominio*. Para la elaboración de la ontología de *Ánima*, se considerará que un individuo es *la representación de un elemento del dominio, y por tanto, tiene una o varias propiedades que lo definen.*

### Clase

En una ontología, una clase consiste en *un conjunto de individuos que comparten las mismas propiedades y/o comportamientos*, tal y como se ha mencionado en el apartado 5.1.3 de la sección 5.1.

### Propiedades

La *Real Academia Española* [71] define una propiedad como *"Atributo o cualidad esencial de alguien o algo"*. Según indica la referencia del *OWL Web Ontology Language Reference* [72], *OWL* distingue dos categorías principales de propiedades:

- ***Datatype properties***: Las propiedades de tipos de datos conectan individuos con valores de datos.
- ***Object properties***: Las propiedades de objeto conectan individuos con otros individuos.

### Anotaciones

Como definen Parsia y Kalyanpur [73], una anotación es una propiedad que se puede aplicar de manera uniforme a todos los tipos de entidad de *OWL*: ontologías, clases, individuos y propiedades. Esta propiedad se considera como un comentario, y aunque no pueden definir axiomas, permiten añadir información útil a las entidades pertenecientes a cualquier ontología.

Las anotaciones son un elemento fundamental para el desarrollo de este proyecto, pues posibilitan la introducción de información necesaria para algunas entidades de las ontologías, que no pueden o no deben ser indicadas como propiedades de las mismas, pues no forman parte de su descripción, sino que aportan información imprescindible para el correcto funcionamiento de la aplicación.

En este proyecto se utilizan diversos tipos de anotaciones, que se podrían separar en dos conjuntos:

- ***Standard annotations***: Consideramos anotaciones estándar a aquellas que permiten introducir información extra a cualquier entidad de la ontología, incluyendo a esta última también. Son anotaciones que están definidas previamente. Este conjunto esta formado por las siguientes anotaciones:

- *rdfs:comment*: Esta anotación se utilizará como descripción de las entidades a las que se vincula, por lo que deberá contener la referencia completa de información del elemento al que hace alusión. El tipo de dato asociado debe ser **xsd:string**.
- *rdfs:IsDefinedBy*: Esta anotación permite indicar qué elemento indica o define el valor de la entidad a la que está asociada. Sólo debe contener el nombre de dicho elemento con el tipo de dato **xsd:string**.

- ***Stage annotations***: Las anotaciones de etapa son anotaciones definidas por el equipo de desarrollo para poder aportar la información necesaria para determinar qué entidades se pueden considerar como *etapas de creación*, y de qué manera deben ser procesadas. Toda ontología que pertenezca a este proyecto deberá disponer de ellas adecuadamente para definir las etapas de creación de un personaje. A continuación se indica cuales son:

  - *CreationScheme*: Esta anotación describe cuál es el esquema de creación de un personaje. Debe contener de manera ordenada las diversas etapas por las que debe pasar el usuario para crear su personaje. Esta anotación debe ubicarse en cada individuo de la etapa raíz del proceso de creación, utilizando el tipo de dato **xsd:string**.
  - *CreationSchemeRoot*: Esta anotación indica qué clase representa la etapa raíz del proceso de creación. Debe estar ubicada en el individuo de la clase ***ClassDefinition*** correspondiente a la etapa raíz del proceso de creación, y debe tener asociado el valor booleano **true**.
  - *GeneralCostDefinedBy*: Esta anotación permite indicar qué elemento define el valor máximo de puntos de una o varias etapas. Debe ser asociada al individuo de la clase ***ClassDefinition*** correspondiente a la etapa de creación que así lo requiera. Se utiliza de la misma forma que la anotación *rdfs:IsDefinedBy*.
  - *SubstageOrder*: Esta anotación indica a las etapas que lo requieran, en qué posición se debe mostrar. Normalmente se utiliza en etapas que formen parte de una etapa más general, siendo aplicada al individuo de la clase ***ClassDefinition*** correspondiente. Su valor debe ser númerico y estar acompañado del tipo de datos **xsd:unsignedInt**
  - *ValuedListInfo*: Esta anotación debe contener toda la información necesaria para que una etapa de creación que requiera de la vista de introducción de valores realice los cálculos de la manera adecuada. Debe estar asociada al individuo de la clase ***ClassDefinition*** correspondiente, y debe utilizar el tipo de datos **xsd:string**.
  - *ViewType*: Esta anotación indica el tipo de vista que debe mostrar la interfaz gráfica de la aplicación cuando se muestre la etapa a la que está vinculada. Debe estar asociada al individuo de la clase ***ClassDefinition*** correspondiente, y debe indicar el nombre exacto del tipo de vista utilizando el tipo de datos **xsd:string**.

### Etapas de creación

Uno de los objetivos del proyecto es que a la hora de crear el personaje, el usuario pueda conocer de forma clara los pasos que requiere el juego para crear el personaje de la forma más completa posible. A estos pasos los llamaremos *etapas de creación*.

Para poder definir las etapas de creación de *Ánima*, se han planteado las siguientes preguntas:

1. ¿Cuáles son las etapas de creación de personajes del *Ánima*?
2. ¿Se pueden dividir esas etapas en subetapas más pequeñas?
3. ¿Cuál debe ser la etapa inicial del proceso de creación?
4. ¿Son necesarias todas las etapas de creación para cada personaje?
5. ¿Qué tipo de interacción debe tener el usuario con cada etapa?

La primera respuesta suele encontrarse en los libros de los juegos de rol, ya que es una información fundamental para poder crear los personajes. Así sucede en el caso de *Ánima*, de manera que echando un ojo rápido al libro, podemos saber que las etapas de creación de un personaje en *Ánima* son las siguientes:

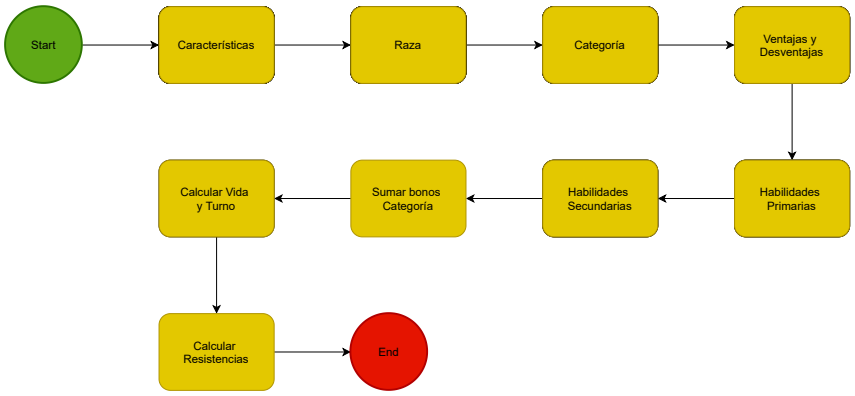


Figura 6.2: Etapas de creación de *Ánima: Beyond Fantasy*

Aunque el proceso aparenta ser rápido y sencillo en un principio, cabe recordar que *Ánima* es conocido entre los jugadores de rol por su gran complejidad en la creación de personajes, llegando al punto de frustrar a los jugadores debido a la inmensidad de información necesaria para completar el personaje. En consecuencia, se ha considerado inviable abordar el proceso de creación de forma directa, y se ha planteado modular el proceso en un mayor número de etapas, que sean más sencillas e intuitivas. Así, el equipo de desarrollo considera **preferible un proceso largo de etapas sencillas** a un proceso corto de etapas complejas.

Después de investigar sobre el juego en profundidad, el equipo de desarrollo ha desglosado las etapas de creación de personajes de *Ánima* en subetapas que pueden ser mostradas en una única vista y que facilitan una interacción intuitiva por parte de los usuarios, y por consiguiente, serán las **etapas de creación** de nuestra aplicación.

Durante la investigación sobre el juego se observa que hay diferentes tipos de personajes, definidos por reglas y que presentan una lista de propiedades y habilidades a las que pueden tener acceso. En *Ánima*, a estos tipos se les conoce como *Categorías*. En consecuencia, la *Categoría* de un personaje determinará las etapas de creación del mismo y es por esto que el primer elemento del proceso deberá ser la elección de *Categoría*.

Una vez conocemos el orden y las etapas de creación de un personaje, falta saber cómo puede interactuar el usuario con cada etapa, pues cada una de ellas puede requerir un tipo de acción diferente por parte del usuario. Además, cada juego dispone de etapas diferentes, lo que hace inviable diseñar una vista para la interacción de cada etapa de cada juego.

La propuesta del equipo de desarrollo para salvar este obstáculo consiste en **generalizar los tipos de interacción** que pueden requerir las etapas de creación de cualquier juego. Esto permite diseñar sólo una vista por cada tipo de interacción, y posteriormente vincular cada etapa con el tipo de interacción más adecuado. En el caso de *Ánima*, se han contemplado los siguientes tipos de interacción:

- ***Selección única***: El usuario sólo necesita elegir una opción de todas las disponibles.
- ***Selección múltiple***: El usuario puede elegir varias opciones de todas las disponibles. Este caso tiene varias posibilidades:
  - *Coste estático*: Todas las opciones tienen el mismo coste, y en caso de estar agrupados, no es necesario realizar gasto alguno de puntos para poder seleccionar elementos de un grupo.
  - *Coste estático y de grupo*: Todos las opciones tienen el mismo coste, y en caso de estar agrupados, poder optar a seleccionar elementos de un grupo hay que pagar un coste.
  - *Coste dinámico*: Cada opción tiene un coste diferente.
- ***Introducción de valores***: El usuario debe introducir valores en los elementos mostrados.

Después de apreciar estos posibles casos, se han revisado otros posibles *RPGs* para comprobar que los tipos de interacción previamente indicados son reutilizables para otros juegos, y por tanto, convenientes para el desarrollo del presente proyecto.

# CAPÍTULO 7

## Construcción del Sistema

# CAPÍTULO 8

## Pruebas del Sistema

# Parte III

# Epílogo

CAPÍTULO 9

# Manual de Implantación y Explotación

CAPÍTULO 10

# Manual de Usuario

CAPÍTULO 11

# Manual de Desarrollador

CAPÍTULO 12

# Conclusiones

## 12.1. Bibliografía

# CAPÍTULO 13

## Bibliografía


[1] E Adams. «Fundamentals of game design». En: *Choice Rev. Online* 47.08 (2010), págs. 47–4462–47-4462. ISSN: 0009-4978. DOI: 10.5860/choice.47-4462. URL: https://books.google.es/books?hl=es%7B%5C&%7Dlr=%7B%5C&%7Did=1vQqAwAAQBAJ%7B%5C&%7Doi=fnd%7B%5C&%7Dpg=PT5%7B%5C&%7Ddq=Fundamentals+of+Role-Playing+Game+Design%7B%5C&%7Dots=4vPCfiy19v%7B%5C&%7Dsig=G74otw1TBCWnxm8J4Y6YrqbmtC0.

[2] Pedro Ramos Villagrasa y Manuel Sueiro Abad. «Personalidad y elección de personaje en los juegos de rol: dime quién eres y te diré quién prefieres ser». En: *Educ. Knowl. Soc.* 11.3 (2010), págs. 8-26. ISSN: 1138-9737. URL: https://www.redalyc.org/pdf/2010/201021093002.pdf.

[3] Susanne Isaksson. «Character Creation Processes in MMORPGs -A qualitative study of determining important factors». En: (2012). URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2:543179.

[4] Lajos Egri. *The Art of Dramatic Writing: Its Basis in the Creative Interpretation of Human Motives. With an Introd. by Gilbert Miller*. 1960.

[5] Petri Lankoski, Satu Heliö e Inger Ekman. «Characters in Computer Games: Toward Understanding Interpretation and Design». En: *DiGRA '03 - Proc. 2003 DiGRA Int. Conf. Lev. Up* (2003), págs. 1-12. URL: http://www.digra.org/wp-content/uploads/digital-library/05087.10012.pdf.

[6] Anders Tychsen, Michael Hitchens y Thea Brolund. «Character play - The use of game characters in multi-player role-playing games across platforms». En: *Comput. Entertain.* 6.2 (jul. de 2008). ISSN: 15443574. DOI: 10.1145/1371216.1371225. URL: http://doi.acm.org/10.1145/1371216.1371225.

[7] Varios Autores. *Generadores de personajes online — Ars Rolica*. URL: https://arsrolica.wordpress.com/2018/12/19/generadores-de-personajes-online/ (visitado 26-06-2020).

[8] Julián Pérez Porto y María Merino. *Definición de Android - Qué es, Significado y Concepto*. 2015. URL: https://definicion.de/android/ (visitado 29-06-2020).

[9] Peter Golde Anders Hejlsberg, Scott Wiltamuth. *The C# Programming Language - Anders Hejlsberg, Scott Wiltamuth, Peter Golde - Google Libros*. 2003. URL: https://books.google.es/books?hl=es%7B%5C&%7Dlr=%7B%5C&%7Did=ICe7ea4RscUC%7B%5C&%7Doi=fnd%7B%5C&%7Dpg=PT14%7B%5C&%7Ddq=c%7B%5C%7D23+definition%7B%5C&%7Dots=YbRMHVEciW%7B%5C&%7Dsig=ubbZnQxhuG1kxQd6xcLQjVSOb78%7B%5C#%7Dv=snippet%7B%5C&%7Dq=c%7B%5C%7D23%20%7B%5C&%7Df=false%20https://books.google.es/books?hl=es%7B%5C&%7Dlr=%7B%5C&%7Did=6L1Rm031qCkC%7B%5C&%7Doi=fnd%7B%5C&%7Dpg=PA3%7B%5C&%7Ddq=c%7B%5C%7D23%7B%5C&%7Dots=5wOX7GQ1yK (visitado 29-06-2020).

[10] Matthew McCullough Jon Loeliger. *Version Control with Git: Powerful tools and techniques for collaborative ... - Jon Loeliger, Matthew McCullough - Google Libros*. 2012. URL: https://books.google.es/books?hl=es%7B%5C&%7Dlr=%7B%5C&%7Did=aM7-Oxo3qdQC%7B%5C&%7Doi=fnd%7B%5C&%7Dpg=PR3%7B%5C&%7Ddq=git+svn%7B%5C&%7Dots=39CfLEWhxa%7B%5C&%7Dsig=H0H3ujbnqorC25sxQaZfTNZOP5E%7B%5C#%7Dv=onepage%7B%5C&%7Dq=git%20svn%7B%5C&%7Df=false%20https://books.google.com.bo/books?id=qIucp61eqAwC%7B%5C&%7Dprintsec=frontcover%7B%5C&%7Ddq=git%7B%5C&%7Dhl=es-419%7B%5C&%7Dsa=X%7B%5C&%7Dved=0ah (visitado 29-06-2020).

[11] Laura Dabbish y col. «Social coding in GitHub: Transparency and collaboration in an open software repository». En: *Proc. ACM Conf. Comput. Support. Coop. Work. CSCW*. 2012, págs. 1277-1286. ISBN: 9781450310864. DOI: 10.1145/2145204.2145396. URL: http://git-scm.com/.

[12] María Dolores; Lozano Pérez. *Ingeniería del software y bases de datos: tendencias actuales - María Dolores Lozano Pérez - Google Libros*. 2000. URL: https://books.google.com.ar/books?id=bNDzMt6dwNsC%7B%5C&%7Dlpg=PA78%7B%5C&%7Ddq=%7B%5C%7D22Entorno%20de%20desarrollo%20integrado%7B%5C%7D22%7B%5C&%7Dpg=PA78%7B%5C#%7Dv=onepage%7B%5C&%7Dq%7B%5C&%7Df=false (visitado 29-06-2020).

[13] Yohn Daniel Amaya Balaguera. «Metodologías ágiles en el desarrollo de aplicaciones para dispositivos móviles. Estado actual». En: *Rev. Tecnol.* 12.2 (2015). ISSN: 1692-1399. DOI: 10.18270/rt.v12i2.1291. URL: https://revistas.unbosque.edu.co/index.php/RevTec/article/view/1291.

[14] *Patrón Model-View-ViewModel - Xamarin — Microsoft Docs*. URL: https://docs.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm (visitado 29-06-2020).

[15] Thomas R. Gruber. «Toward principles for the design of ontologies used for knowledge sharing». En: *Int. J. Hum. - Comput. Stud.* 43.5-6 (1995), págs. 907-928. ISSN: 10959300. DOI: 10.1006/ijhc.1995.1081. URL: https://www.sciencedirect.com/science/article/pii/S1071581985710816.

[16] Nicola Guarino, Daniel Oberle y Steffen Staab. *Handbook on Ontologies*. 2009. DOI: 10.1007/978-3-540-92673-3.

[17] S Bechhofer y col. *OWL Web Ontology Language Reference*. 2004. URL: http://www.w3.org/TR/2004/REC-owl-ref-20040210/%20http://www.w3.org/TR/owl-ref/.

[18] John H Gennari y col. «The evolution of Protégé: An environment for knowledge-based systems development». En: *Int. J. Hum. Comput. Stud.* 58.1 (2003), págs. 89-123. ISSN: 10715819. DOI: 10.1016/S1071-5819(02)00127-1. URL: https://www.sciencedirect.com/science/article/pii/S1071581902001271.

[19] O Lassila y R R Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. Inf. téc. 19990222. 1999. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.6030.

[20] Zoi Kaoudi, Iris Miliaraki y Manolis Koubarakis. «RDFS reasoning and query answering on top of DHTs». En: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*. Vol. 5318 LNCS. 2008, págs. 499-516. ISBN: 3540885633. DOI: 10.1007/978-3-540-88564-1-32. URL: http://atlas.di.uoa.gr.

[21] Marco De Salvo. *RDFSharp Reference Guide (v2.13)*. 41. URL: https://github.com/mdesalvo/RDFSharp/releases/download/v2.13.0/RDFSharp-2.13.0.pdf.

[22] Carmen Lasa Gómez Alonso Álvarez García, Rafael de las Heras del Dedo. *Métodos Ágiles y Scrum*. 2012, pág. 352. ISBN: 978-84-415-3104-8.

[23] Evren Sirin y Bijan Parsia. «SPARQL-DL: SPARQL query for OWL-DL». En: *CEUR Workshop Proc.* Vol. 258. 2007. URL: http://www.w3.org/Submission/RDQL/.

[24] Geoffrey Sparks. «Una Introducción al UML». En: (2008), págs. 1-47. URL: www.sparxsystems.com.ar-www.sparxsystems.cl.

[25] *Microsoft Visual Studio - Wikipedia, la enciclopedia libre*. URL: https://es.wikipedia.org/wiki/Microsoft%7B%5C_%7DVisual%7B%5C_%7DStudio (visitado 30-06-2020).

[26] *¿Qué es Xamarin? - Xamarin — Microsoft Docs*. URL: https://docs.microsoft.com/es-es/xamarin/get-started/what-is-xamarin (visitado 30-06-2020).

[27] *¿Qué es Xamarin.Forms? - Xamarin — Microsoft Docs*. URL: https://docs.microsoft.com/es-es/xamarin/get-started/what-is-xamarin-forms (visitado 30-06-2020).

[28] Norman Walsh. *A Technical Introduction to XML What is XML?* Inf. téc. URL: https://www.math.unipd.it/%7B~%7Dbasidati/docs/introduction.pdf.

[29] Lori MacVittie, A. *XAML in a Nutshell - Lori MacVittie - Google Knjige*. 2006, pág. 284. ISBN: 0-596-52673-3. URL: https://books.google.es/books?hl=es%7B%5C&%7Dlr=%7B%5C&%7Did=v03elGOy9ogC%7B%5C&%7Doi=fnd%7B%5C&%7Dpg=PT4%7B%5C&%7Ddq=Xaml%7B%5C&%7Dots=WPWDOYfrfS%7B%5C&%7Dsig=EOJ9CZ%7B%5C_%7DFC9k2gizr4pq8n2oGmcI%7B%5C#%7Dv=onepage%7B%5C&%7Dq=Xaml%7B%5C&%7Df=false%20https://books.google.si/books?hl=sl%7B%5C&%7Dlr=%7B%5C&%7Did=v03elGOy9ogC%7B%5C&%7Doi=fnd%7B%5C&%7Dpg=PT4%7B%5C&%7Ddq=xaml%7B%5C&%7Dots=WOUCO07ogY%7B%5C&%7Dsig=Fgs08I%7B%5C_%7DX.

[30] James Hendler, Tim Berners-Lee y Eric Miller. *Integrating Applications on the Semantic Web*. 2002. URL: https://www.w3.org/2002/07/swint (visitado 30-06-2020).

[31] Mary Shaw y David Garlan. «Formulations and formalisms in software architecture». En: vol. 1000. 1995, págs. 307-323. DOI: 10.1007/bfb0015251.

[32] Gaston Addati. *Escuela Superior Técnica Facultad de Ingeniería*. Inf. téc. 2013, pág. 132. URL: www.econstor.eu.

[33] A. Pérez, J. & Gardey. *Definición de modelo de datos - Qué es, Significado y Concepto*. 2012. URL: https://definicion.de/modelo-de-datos/ (visitado 05-07-2020).

[34] A Muñoz, J Aguilar - Avances en Sistemas e Informática y Undefined 2009. «Ontología para bases de datos orientadas a objetos y multimedia». En: *Rev. Av. en Sist. e Informática* 6.2 (2009), págs. 167-184. ISSN: 1657-7663. URL: https://revistas.unal.edu.co/index.php/avances/article/view/20372.

[35] Benson Moyo y col. «Empirical evaluation of software development methodology selection consistency : A case study using Analytical Hierarchy Process». En: *search.proquest.com* (). URL: http://search.proquest.com/openview/3df28c8b722e916cac59dabc6be8b03d/1?pq-origsite=gscholar%7B%5C&%7Dcbl=1976341.

[36] RS Pressman y JM Troya. «Ingeniería del software». En: (1988). URL: http://fondoeditorial.uneg.edu.ve/citeg/numeros/c02/c02%7B%5C_%7Dart10.pdf.

[37] Carlos Gerardo Prieto Álvarez. «Adaptación de las Metodologías Tradicionales Cascada y Espiral para la Inclusión de Evaluación Inicial de Usabilidad en el Desarrollo de Productos de Software en México.» En: (2013).

[38] Digital Guide IONOS. *El modelo en cascada en el desarrollo de software -*. 2019. URL: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-modelo-en-cascada/ (visitado 09-07-2020).

[39] *Modelo En Espiral, Los Sistemas De Ciclo De Vida De Desarrollo, Proceso De Desarrollo De Software imagen png - imagen transparente descarga gratuita*. URL: https://www.freepng.es/png-zfl949/ (visitado 09-07-2020).

[40] Ramon Ventura Roque Hernandez y col. «Comparación empírica entre el Proceso Unificado y el desarrollo de software por Prototipos». En: *West. Hemispheric Trade Conf.* 2015, págs. 235-244. ISBN: 0123566789. URL: http://www.academia.edu/download/54580094/19Conf-Sessions.pdf%7B%5C#%7Dpage=245.

[41] *Modelo de Prototipos - Portafolio de Evidencia Erika.* URL: https://sites.google.com/site/portafoliodeevidenciaerika/tema-3---desa/modelo-de-prototipos (visitado 02-07-2020).

[42] Johanna Patricia Zumba. «Evolución de las Metodologías y Modelos utilizados en el Desarrollo de Software». En: *INNOVA Res. J.* 3.10 (2018), págs. 20-33. ISSN: 2477-9024. DOI: 10.33890/innova.v3.n10.2018.651. URL: https://dialnet.unirioja.es/servlet/articulo?codigo=6777227%7B%5C&%7Dinfo=resumen%7B%5C&%7Didioma=SPA%20https://dialnet.unirioja.es/servlet/articulo?codigo=6777227%7B%5C&%7Dinfo=resumen%7B%5C&%7Didioma=ENG%20https://dialnet.unirioja.es/servlet/articulo?codigo=6777227.

[43] Alejandro Aldás Alarcón. *Modelo de proceso incremental Fuente:... — Download Scientific Diagram.* 2016. URL: https://www.researchgate.net/figure/Figura-10-Modelo-de-proceso-incremental-Fuente%7B%5C_%7Dfig6%7B%5C_%7D326571456 (visitado 02-07-2020).

[44] Kent Beck y Et.al. *Principios del Manifiesto Ágil.* 2001. URL: https://agilemanifesto.org/iso/es/principles.html (visitado 02-07-2020).

[45] Matt Stephens y Doug Rosenberg. *Extreme Programming Refactored: The Case Against XP.* 2003. DOI: 10.1007/978-1-4302-0810-5.

[46] K Schwaber y Jeff Sutherland. «The scrum guide suomeksi». En: *Scrum. org, Oct.* 2.July (2011), pág. 17. ISSN: 00195847. DOI: 10.1053/j.jrn.2009.08.012. URL: http://www.scrum.org/scrumguides/%7B%5C%%7D5Cnhttp://pdf4420.psxbook.com/scrum%7B%5C_%7D1868546.pdf.

[47] Alistair Cockburn. *Crystal Clear: A Human-Powered Methodology for Small Teams (The Agile Software Development Series).* 2004. URL: https://www.researchgate.net/publication/234820806%20http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20%7B%5C&%7Dpath=ASIN/0201699478.

[48] César Rodríguez y Rubén Dorado. *Porque implementar Scrum?* Inf. téc. URL: https://journal.universidadean.edu.co/index.php/Revistao/article/view/1253.

[49] *Visual Studio Code Vector Logo - Download Free SVG Icon — Worldvectorlogo.* URL: https://worldvectorlogo.com/es/logo/scrumorg-1%20https://worldvectorlogo.com/logo/visual-studio-code (visitado 07-07-2020).

[50] Manuel Trigás Gallego. «Metodología Scrum». En: *Gest. Proy. informáticos* (2012), pág. 56. URL: https://s3.amazonaws.com/academia.edu.documents/39164786/mtrigasTFC0612memoria%7B%5C_%7D1.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A%7B%5C&%7DExpires=1554050320%7B%5C&%7DSignature=lv5tLOeYpbqQj0Q14ZHSM3k12H8%7B%5C%%7D3D%7B%5C&%7Dresponse-content-disposition=inline%7B%5C%%7D3B%20filename%7B%5C%%7D3DMtrigas%7B%5C_%7DTFC0612memoria%7B%5C_%7D1.

[51] *Scrum: qué es y cómo puede salvar tu proyecto — Herizont*. URL: https://www.herizont.com/2019/03/07/metodologia-agil-scrum/ (visitado 07-07-2020).

[52] «El rol del business analyst en entornos agile». En: (). URL: https://www.netmind.es/knowledge-center/el-rol-del-business-analyst-en-entornos-agile/.

[53] *Cuál es la diferencia entre un Enterprise Architect, un Solutions Architect y un Technical Architect*. URL: https://www.linkedin.com/pulse/cu%7B%5C'%7Ba%7D%7Dl-es-la-diferencia-entre-un-enterprise-architect-solutions-mu%7B%5C~%7Bn%7D%7Doz/?originalSubdomain=es (visitado 13-07-2020).

[54] *Qué es un UX Designer y cuáles son sus roles — Blog de Marketing Digital — Making Science*. URL: https://www.makingscience.com/blog/que-es-un-ux-designer-y-cuales-son-sus-roles/ (visitado 13-07-2020).

[55] *Roles desarrollo del software*. URL: https://es.slideshare.net/SebastianRamrez2/roles-desarrollo-del-software (visitado 13-07-2020).

[56] *What are the roles and responsibilities of a Tester?* URL: http://tryqa.com/what-are-the-roles-and-responsibilities-of-a-tester/ (visitado 13-07-2020).

[57] *Documentalist - Wikipedia*. URL: https://en.wikipedia.org/wiki/Documentalist (visitado 13-07-2020).

[58] JOAO MARTINS FERREIRA MIRANDA DE Távora. «Tesis de máster». Tesis doct. 2014, pág. 88.

[59] K Wiegers y J Beatty. «Software requirements». En: (2013). URL: https://books.google.es/books?hl=es%7B%5C&%7Dlr=%7B%5C&%7Did=nbpCAwAAQBAJ%7B%5C&%7Doi=fnd%7B%5C&%7Dpg=PT32%7B%5C&%7Ddq=Wiegers,+Karl+E.+(2003).+Software+Requirements+2:+%7B%5C&%7Dots=9oSXHW7yMn%7B%5C&%7Dsig=AiN5Z3h2WPJ9H1jqPtwZbhb6-oU.

[60] A Stellman y J Greene. «Applied software project management». En: (2005). URL: https://books.google.es/books?hl=es%7B%5C&%7Dlr=%7B%5C&%7Did=IYdJocLVa8wC%7B%5C&%7Doi=fnd%7B%5C&%7Dpg=PR1%7B%5C&%7Ddq=Stellman,+Andrew%7B%5C%%7D3B+Greene,+Jennifer+(2005).+Applied+Software+Project+Management.+O%7B%5C%%7D27Reilly+Media.+%7B%5C&%7Dots=zZd5fiXk6k%7B%5C&%7Dsig=NCZ98VWzb-mGmy%7B%5C_%7DDG1pJ3rvYMGA.

[61] Lidia Fuentes y M Troya. «Lección 1 Desarrollo de Software Basado en Componentes». En: (), págs. 1-22.

[62] Android Studio. *Introducción a Android Studio — Desarrolladores de Android*. 2019. URL: https://developer.android.com/studio/intro?hl=es-419%20https://developer.android.com/studio/intro%7B%5C%%7D0Ahttps://developer.android.com/studio/intro?hl=es-419 (visitado 15-07-2020).

[63] *React Native - Wikipedia*. URL: https://en.wikipedia.org/wiki/React%7B%5C_%7DNative (visitado 15-07-2020).

[64] Stewart Robinson y col. «Conceptual modeling: Definition, purpose and benefits». En: *Proc. - Winter Simul. Conf.* Vol. 2016-Febru. 2016, págs. 2812-2826. ISBN: 9781467397438. DOI: 10.1109/WSC.2015.7408386.

[65] Harpreet Kaur y Pardeep Singh. «Unified Modeling Language: Standard language for foftware architecture development». En: *Syst. Archit. Creat. Build. complex Syst.* 15 (2015), págs. 3-9. URL: https://www.researchgate.net/publication/266869058.

[66] Javier Ortega. «Transformación de ontologías OWL a UML para la indexación y recuperación mediante esquema basado en grafos.» En: *Univ. Carlos III Madrid.* (2015).

[67] Conrad Bock y col. «OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax (Second Edition)». En: *Online* (2012), págs. 1-133. URL: https://www.w3.org/TR/owl2-syntax/%7B%5C#%7DIndividuals.

[68] Alistair Cockburn. «Humans and Technology in preparation for Addison-Wesley Longman». En: 3 (2000), pág. 204. URL: https://www.infor.uva.es/%7B~%7Dmlaguna/is1/materiales/BookDraft1.pdf.

[69] *Building Cross-Platform Reactive .NET Apps — by Artyom V. Gorchakov — Medium*. URL: https://medium.com/@worldbeater/reactive-ui-fody-cross-platform-forms-7b501d79f46b (visitado 24-07-2020).

[70] Tian Lou. «A comparison of Android Native App Architecture MVC, MVP and MVVM». En: *Research.Tue.Nl* (2016), pág. 45. URL: https://research.tue.nl/files/48628529/Lou%7B%5C_%7D2016.pdf.

[71] *propiedad — Definición — Diccionario de la lengua española — RAE - ASALE*. URL: https://dle.rae.es/propiedad (visitado 28-07-2020).

[72] S Bechhofer y col. *OWL Web Ontology Language Reference*. 2004. URL: https://www.w3.org/TR/owl-ref/%7B%5C#%7DProperty%20http://www.w3.org/TR/owl-ref/.

[73] Bijan Parsia y Aditya Kalyanpur. «Annotating OWL ontologies». En: *CEUR Workshop Proc.* Vol. 184. 2004, págs. 125-126. URL: http://www.w3.org/TR/owl-ref/%7B%5C#%7DHeader.

CAPÍTULO 14

# Información sobre Licencia